Optimization of the Processing of Data Streams on Roughly Characterized Distributed Resources

Daniel Millot and Christian Parrot, *Member, IEEE*

Abstract—The AS4DR (Adaptive Scheduling for Distributed Resources) scheduling method presented in this paper aims at maximizing throughput, when processing several data streams by divisible load applications on star-shaped distributed memory platforms, with available speeds for communicating and computing which may be poorly estimated, or varying over time. The total workload is supposed to be unknown. According to the computation cost model, AS4DR can either maximize throughput, or CPU utilization by avoiding data-starvation of the computing units. An experimental assessment of the adaptation of the workload distribution to the variation of the communicating and computing speeds has been performed that shows that the use of AS4DR can significantly improve the throughput. This paper also experimentally assesses a resource selection method to set up star-shaped clusters of distributed resources, so as to process efficiently a set of data streams with AS4DR.

Index Terms—Parallel application, distributed memory, divisible load scheduling, adaptive scheduling, data-intensive application

1 Introduction

The continuously increasing flood of data, coming from results of biological and physical experiments or from sensors, is usually processed by large, but heterogeneous [1], [2] and often only roughly characterized platforms. Many of the data-intensive applications involved in this context are divisible load applications [3]: the workload can be divided into chunks as small as necessary for the scheduling and each chunk can be processed independently from the others. Example applications are search for a pattern in text or in DNA sequences as well as video streams encoding [4], [5].

Let us consider a set of similar continuous data streams of unlimited throughputs to be processed by a divisible load application, on a collection of heterogeneous star-shaped platforms. As the data streams are supposed to be similar, each of them is arbitrarily associated to one platform. One node in each platform acquires the data stream allocated to the platform it belongs to, and splits this stream into chunks it distributes to the other nodes of the platform. Each of these nodes processes the load it receives and sends the result back, just as for many data parallel applications [6]. According to the master-workers paradigm, the processing units which distribute the load act as masters and the computation units which do the processing act as workers [7]. Besides, we assume that the computation units have an unlimited buffering capability, unlike [8].

According to the size of the chunks, communication complies either with the multi-port communication model, which allows broadcasting, or with the one-port communication model, for which only one communication from (or to) one node can be performed at the same time by a node. In the latter case, there is a risk of contention when workers compete to access a master; for instance, when trying to broadcast large messages with classical implementations of MPI [9]. We suppose links are bidirectional and computation can overlap communication. As usual when one wants to distribute computing efficiently, let us assume that the communication speed between a worker and its master is high enough compared to the computation speed of this worker, which depends on both the available CPU frequency and the computation algorithmic complexity. Hereafter, the available speeds for communication and computation will be called execution parameters.

We call "round" a sequence of consecutive actions leading a master to feed all its associated workers once, and to collect the corresponding results. As the data acquisition may last a long time, the processing has to begin before the very last data item is acquired by one of the masters, hence before the total workload is known. Making this assumption has two consequences. First, the minimization of the makespan can no longer be considered as an objective of the scheduler (until the very last data item is acquired by one of the masters). And, second, the scheduling must proceed iteratively, as the workload flows in; the scheduling has to be multi-round.

The processing of each data stream can be split into three phases. The start-up phase begins when the master, associated to a stream, starts to send the very first chunk to a worker of the considered master/workers cluster, and ends when each of these workers has received a chunk to process. Then begins the streaming phase which ends when the last data item in the stream has been acquired by the master. Eventually, the cleanup phase begins and it lasts until the master has received the very last result of processing from each worker.

When the cleanup phase begins, the remaining total workload is known and the makespan of its processing can be minimized, by making the workers complete their work

• *The authors are with the Institut Mines-Telecom, Telecom SudParis, Évry, France. E-mail: {daniel.millot, christian.parrot}@telecom-sudparis.eu.*

Manuscript received 11 Sept. 2014; revised 28 May 2015; accepted 13 June 2015. Date of publication 18 June 2015; date of current version 13 Apr. 2016. Recommended for acceptance by I. Brandic.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2447515

simultaneously; many scheduling methods have addressed this problem [10], [11] and [12]. Before leaving this widely addressed scheduling problem, let us point out that to reduce the cost of reallocating the chunks between workers as much as possible, in order to balance their remaining load when the cleanup phase begins, it is convenient to upper-bound the workload discrepancy between workers during the streaming phase as much as possible.

Let us now focus on the other two phases: the start-up and the streaming phases. Later on in this paper, we will consider that the start-up duration is negligible compared to the streaming phase duration, unlike [13]. The legitimacy of this assumption increases with the ratio of the total workload to the load distributed at the very first round. Under this assumption, it is reasonable to focus on the streaming phase. AS4DR aims at maximizing the throughput during this period, since the makespan cannot be an objective function. In order to establish some theoretical results, the study is limited to affine cost models for both computation and communication. As this problem is known to be NP-complete [14], when considering affine costs for computation and communication, it can only be dealt with by means of heuristics.

The ultimate goal of AS4DR is to adapt the scheduling of a divisible load application to the lack of information about the execution parameters automatically, in order to maximize the total throughput of processed data. This lack of information can be due both to poor estimates of the execution parameters and to their unpredictable variation, which is called dynamicity in the sequel. To the best of our knowledge, only a few methods can cope with the non-specification of the total workload: DA1 [15], OLMR [16] and AS4DR [17]; all three can deal with the multi-port communication model, whereas only AS4DR prevents contention when considering the one-port communication model.

The rest of the paper is organized as follows. Section 2 presents fundamental results on the AS4DR method when the characteristics of the platform are in a steady state, but with poor awareness. In Section 3, the adaptivity of AS4DR scheduling to this lack of awareness is assessed experimentally, still in a steady context. Afterwards, the behaviour of AS4DR, when the characteristics of the platform vary over time, is empirically studied in Section 4. In the context of Section 4, the same size of data can lead to very different processing and communicating times. Then, in Section 5, the adaptivity of AS4DR scheduling to the dynamicity of the execution parameters is assessed experimentally. In order to focus on the distribution of the load, rather than on the setting up of master/workers clusters of computing units, Sections 2, 3, 4, and 5 consider only one data stream and a single master/workers cluster. Then Section 6 addresses the problem of the resource selection to form the master/workers clusters, one for each data stream at most. Before concluding and giving hints on future work in Section 8, Section 7 is dedicated to related works.

Section 2 has been partially published in [17]. Sections 3 and 5 have been published in [18], [19], whereas Section 6 has been published in [20]. Section 4, which compares the effects of different types of variation of the execution parameters over time, has never been published. All the experiments described in this paper have been conducted with the SimGrid framework [21].

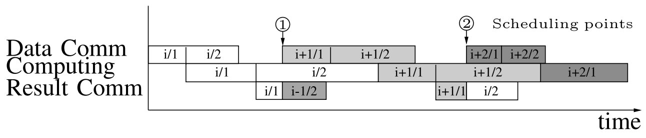


Fig. 1. Overlapping between communication and computation.

2 Presentation of the AS4DR Method

This section presents the AS4DR method, to process a single data stream, when the execution parameters are constant in time (steady context), but possibly poorly estimated. The proof of the four propositions considered in this section can be found in the appendices.

Let us consider a star-shaped platform with a master connected to W workers $(\mathrm{w_i})_{\mathrm{i=0,W-1}}$ by bidirectional links. Let $\alpha_{\mathrm{w,i}}$ be the size of the chunk sent to any worker w, for round i. Let τ and $\sigma_{\mathrm{w,i}}$ be respectively the desired and an estimated time durations between the beginning of the sending of a chunk of size $\alpha_{\mathrm{w,i}}$, and the end of the reception of the corresponding result by the master. As we want computation to overlap communication, $\sigma_{\mathrm{w,i}}$, which is defined in the sequel, doesn't take the communication duration of round i into account.

The basic idea of this multi-round method is to adapt $\alpha_{\mathrm{w,i}}$ according to method DA1 [15]:

$$\alpha_{\mathrm{w,i}} := \alpha_{\mathrm{w,i-1}} \frac{\tau}{\sigma_{\mathrm{w,i-1}}} \qquad \text{for i} > 0. \tag{1}$$

Whereas DA1 does not, AS4DR splits each chunk it has to deliver (to a worker for a round) into two subchunks that it delivers in a row to the worker, as OLMR does. Having sent subchunks of arbitrarily chosen sizes $\dot{\alpha}_{\mathrm{w,0}}$ and $\ddot{\alpha}_{\mathrm{w,0}}$ to each worker w for the first round, the AS4DR scheduler then sends to worker w, for each round i, two subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$ of respective sizes $\dot{\alpha}_{\mathrm{w,i}}$ and $\ddot{\alpha}_{\mathrm{w,i}}$, such that

$$\dot{\alpha}_{\mathrm{w,i}} + \ddot{\alpha}_{\mathrm{w,i}} = \alpha_{\mathrm{w,i}}. \tag{2}$$

Dividing the chunks into two parts is enough to apply the principle of the multi-installment strategy [22], in order to allow the computation to overlap communication between workers and master, as can be seen in Fig. 1, for a single worker. In order to be able to establish the result stated in Proposition 2, let us assume that the ratio, denoted θ_{w}, between $\dot{\alpha}_{\mathrm{w,i}}$ and $\alpha_{\mathrm{w,i}}$ is constant:

$$\theta_{\mathrm{w}} \equiv \frac{\dot{\alpha}_{\mathrm{w,i}}}{\alpha_{\mathrm{w,i}}}. \tag{3}$$

Figs. 2 and 3 give the scheduling algorithm of the AS4DR method (with references to the previous equations, and to others stated further in the text). As can be observed in Fig. 1, for i > 0, round i for worker w is composed of three phases:

- transmission of the data from master to worker, lasting $\dot{\mathrm{D}}_{\mathrm{w,i}}$ and $\ddot{\mathrm{D}}_{\mathrm{w,i}}$ for subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$ respectively,
- worker computation on the received data, lasting $\dot{\mathrm{C}}_{\mathrm{w,i}}$ and $\ddot{\mathrm{C}}_{\mathrm{w,i}}$ for subchunks $\dot{\mathbf{s}}$ and $\ddot{\mathbf{s}}$ respectively,

```
• Set $\tau$ and $(\alpha_{w,0}, \theta_{w,0})_{0 \le w \le W-1}$ ...... equ. (14), (9)
• Set $(\dot{\alpha}_{w,0}, \ddot{\alpha}_{w,0})_{0 \le w \le W-1}$ ......... equ. (3), (2)
• Set $(d_w)_{0 \le w \le W-1}$ .................... equ. (13)
• After the delay $d_w$, post $\dot{s}$ and $\ddot{s}$ data to each worker w
while (the last data item has not been acquired) do
    • Get a $\dot{s}$ result from some worker w
    • Compute the size of the next $\dot{s}$ and $\ddot{s}$
    for worker w .............. equ. (4), (1), (3), (2)
    • After the delay $d_w$, post $\dot{s}$ and $\ddot{s}$ data to each worker w
    • Get previous $\ddot{s}$ result from worker w
end while
```

Fig. 2. AS4DR scheduling: Master.

- transmission of the computation results from worker to master, lasting $\dot{R}_{w,i}$ and $\ddot{R}_{w,i-1}$ for subchunks $\dot{s}$ and $\ddot{s}$ respectively.

It is worth noting in Fig. 1 that the result corresponding to the $\ddot{s}$ subchunk of some round is not returned straight away to the master, but just after the result corresponding to the $\dot{s}$ subchunk of the next round has itself been returned. As the scheduler does not make use of the return of $\ddot{s}$ results in any way, delaying this return does not bring any drawback. On the other hand, communicating in a row these $\dot{s}$ and $\ddot{s}$ results makes easier the prevention of contentions, by helping to forecast the moment when workers send back their $\ddot{s}$ results.

As we suppose that the communication and computation costs are both roundwise affine in the size of the corresponding chunk, we have:

$$\begin{aligned}
\dot{D}_{w,i} &= \theta_w \frac{\alpha_{w,i}}{B_w^D} + b_w^D, & \ddot{D}_{w,i} &= (1-\theta_w)\frac{\alpha_{w,i}}{B_w^D} + b_w^D,\\
\dot{C}_{w,i} &= \theta_w \frac{\alpha_{w,i}}{F_w} + f_w, & \ddot{C}_{w,i} &= (1-\theta_w)\frac{\alpha_{w,i}}{F_w} + f_w,\\
\dot{R}_{w,i} &= \theta_w \frac{\alpha_{w,i}}{B_w^R} + b_w^R, & \ddot{R}_{w,i} &= (1-\theta_w)\frac{\alpha_{w,i}}{B_w^R} + b_w^R.
\end{aligned}$$

Where F_w is the available computation speed (relative to the processing of one data workload unit) for worker w. It is worth recalling that this computation speed depends both on the platform, by means of the clock frequency of worker w, and on the application, by means of the algorithmic complexity. Likewise, B_w^D (resp. B_w^R) is the available communication speed (relative to one data workload unit) of the link from the master to worker w (resp. from worker w to the master). Finally b_w^D, b_w^R and f_w are the respective latencies for a transfer of data from the master to worker w, for a transfer of result from worker w to the master and for a computation on worker w respectively. The affine variation of $\dot{R}_{w,i}$ and $\ddot{R}_{w,i}$ according to the amount of data processed does not compel the size of the result sent back from the worker w to the master to be equal to the size of the data processed by the worker w. When considering an affine cost model for communication, it is sufficient to assume that the size of the result depends on the size of the data in an affine way, to make the definition of $\dot{R}_{w,i}$ and $\ddot{R}_{w,i}$ hold.

```
while (the last subchunk has not been posted) do
    • Get a subchunk data from master
    • Process the subchunk data
    if ($\dot{s}$) then
        • Post $\dot{s}$ and previous $\ddot{s}$ results to master
    end if
end while
```

Fig. 3. AS4DR scheduling: Worker.

Let us recall that $\sigma_{w,i}$ denotes an estimated duration of round i for worker w. From $\dot{C}_{w,i}$, the measured duration of the computation of subchunk $\dot{s}$ at round i by worker w, the value of $\sigma_{w,i}$ can be extrapolated as follows:

$$\sigma_{w,i} \equiv \frac{\dot{C}_{w,i} - f_w}{\theta_w} + 2f_w. \tag{4}$$

$$\text{As a result,} \qquad \sigma_{w,i} = \alpha_{w,i}\frac{1}{F_w} + 2f_w. \tag{5}$$

From (5) and (1), we get

$$\sigma_{w,i} = 2f_w + \tau - \frac{2f_w\tau}{\sigma_{w,i-1}}. \tag{6}$$

This homographic sequence can be defined if, and only if, $\alpha_{w,i}$ and f_w are both assumed not to equal zero simultaneously.

Proposition 1. *Let us assume that communication between master and workers is contention-free. Then the sequence $(\sigma_{w,i})_i$ built according to the AS4DR method converges monotonically and linearly. More, the schedule installed with the AS4DR method is asymptotically stable.*

$$\begin{aligned}
&if\ 2f_w < \tau, then \lim_{i\to+\infty}\sigma_{w,i} = \tau,\\
&\qquad\qquad\qquad \lim_{i\to+\infty}\alpha_{w,i} = F_w(\tau - 2f_w),\\
&if\ \tau \le 2f_w, then \lim_{i\to+\infty}\sigma_{w,i} = 2f_w,\\
&\qquad\qquad\qquad \lim_{i\to+\infty}\alpha_{w,i} = 0.
\end{aligned}$$

Actually, the duration τ, typically targeted for a round, is much larger than that of the computation latency f_w. Therefore, only the first case: $2f_w < \tau$, will be considered in the sequel.

Dates of posting data from the master, as well as dates of getting results by the master, are asymptotically periodic, with the same period τ for all the workers.

From (6) we get: $\frac{\tau-\sigma_{w,i}}{\tau-\sigma_{w,i-1}} = \frac{2f_w}{\sigma_{w,i-1}}$, and we can claim that the smaller f_w, compared to τ, the faster the convergence of the sequence $(\sigma_{w,i})_i$. In other words, the smaller f_w, compared to τ, the faster the adaptation of the AS4DR scheduling to the poorness of the estimate of the execution parameters.

It is worth noting that Proposition 1 is true either if the master serves the workers in a pre-determined order (round-robin) or if it serves the first worker to be idle systematically (FIFO).

The data-starvation of the workers can be responsible for two types of idleness we call: inter-round idleness and intra-round idleness, later in this paper. Figs. 4 and 5

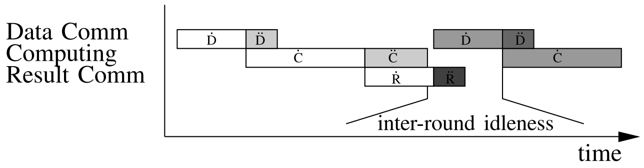


Fig. 4. Example of inter-round idleness.

illustrate these two types of idleness, for a single worker. Usually, the prevention of data-starvation needs a full overlapping of communication with computation. Proposition 2 expresses a refined version of this principle suited to AS4DR, by stating a necessary and sufficient condition for the prevention of workers data-starvation.

Proposition 2. *Let us respectively define, $\forall$ i, $\theta^{min}_{w,i}$ and $\theta^{\max}_{\mathrm{w,i}}$ as follows:*

$$\theta^{min}_{w,i+1} \equiv \frac{D_{w,i+1} - \left(b^D_w + f_w\right)}{D_{w,i+1} + C_{w,i+1} - 2\left(b^D_w + f_w\right)}, \qquad (7)$$

$$\theta^{max}_{w,i+1} \equiv \frac{C_{w,i} - \left(b^D_w + f_w + b^R_w\right)}{D_{w,i+1} + C_{w,i} + R_{w,i} - 2\left(b^D_w + f_w + b^R_w\right)}. \qquad (8)$$

Let us assume that communication between master and workers is contention-free. Then the AS4DR method prevents data-starvation, if and only if, for each worker w, and $\forall$ i

$$\theta^{min}_{w,i+1} \leq \theta_w \leq \theta^{max}_{w,i+1}. \qquad (9)$$

Constraint (9) recalls the necessary constraint for the time spent in communication to be smaller than the one spent in computation, in order to benefit from overlapping.

The throughput for worker w, during round i, that is denoted $\mathrm{t_{w,i}}$ in the sequel, can be define in such a way:

$$\mathrm{t_{w,i}} \equiv \frac{\alpha_{\mathrm{w,i}}}{\dot{\mathrm{C}}_{\mathrm{w,i}} + \ddot{\mathrm{C}}_{\mathrm{w,i}} + \text{time wasted in data-starvation}}.$$

Due to Proposition 2, the value of $\theta_{\mathrm{w,i}}$ can be fixed to avoid data-starvation. As a result,

$$\mathrm{t_{w,i}} = \frac{\alpha_{\mathrm{w,i}}}{\dot{\mathrm{C}}_{\mathrm{w,i}} + \ddot{\mathrm{C}}_{\mathrm{w,i}}}. \qquad (10)$$

Proposition 3. *Let us define*

$$t_w \equiv \left(1 - 2\frac{f_w}{\tau}\right) F_w.$$

Let us assume that communication between master and workers is contention-free. Then the schedule installed with the AS4DR method is asymptotically periodic and

$$lim\ t_{w,i} = t_w.$$

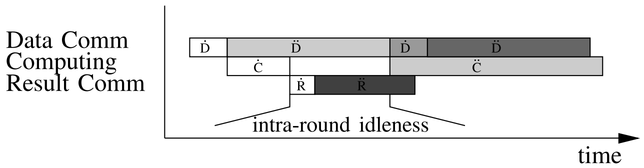


Fig. 5. Example of intra-round idleness.

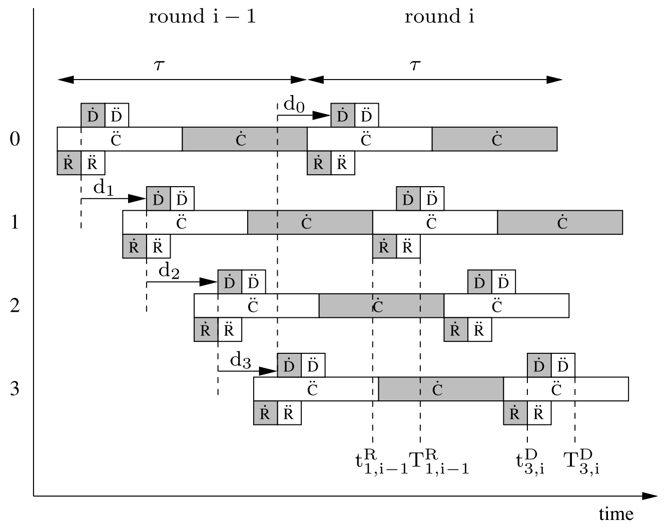


Fig. 6. Contention-free asymptotic schedule.

When considering a linear cost model for computation (when $f_w = 0$), the schedule installed with AS4DR is asymptotically optimal.

Whereas the lack of contention hypothesis holds naturally with a multi-port communication model, the problem of contention avoidance needs to be addressed with a one-port model, to take advantage of Propositions 1, 2 and 3.

To make the instant each worker accesses to the master far enough from the instants the others access too, time delays $\mathrm{d_w}$ are introduced before posting subchunks $\dot{\mathrm{s}}$ to each worker w. Fig. 6 illustrates the model of asymptotic τ-periodic (thus round-robin) scheduling we want to establish, in the case of a four workers platform.

Proposition 4. *The AS4DR method prevents contentions, if and only if, $\forall$ i, for each worker w (modulo W):*

$$d_w \geq \Delta_{w,i};\ where \qquad (11)$$

$\Delta_{w,i} \equiv max\left(\dot{D}_{w-1,i} + \ddot{D}_{w-1,i}, \ddot{R}_{w-1,i-2} + \dot{R}_{w,i-1}\right)$, *with* $\ddot{R}_{w,j} = \dot{R}_{w,j} = 0$ *when* $j < 0$.

The next Proposition states a sufficient condition to prevent contention, when scheduling with AS4DR.

Proposition 5. *Let us assume that, for each worker w (modulo W),*

$$\alpha_{w,0} \geq F_w(\tau - 2f_w). \qquad (12)$$

Let us define

$$d_w \equiv max\left(\dot{D}_{w-1,0} + \ddot{D}_{w-1,0}, \ddot{R}_{w-1,0} + \dot{R}_{w,0}\right). \qquad (13)$$

Then the AS4DR method prevents contentions.

Unfortunately, the values of $\mathrm{F_w}$ and $\mathrm{f_w}$ are only approximately known. In order to find a value of $\alpha_{\mathrm{w,0}}$ such that inequality (12) holds, let us first consider $\overline{\mathrm{F}}_{\mathrm{w}}$ an approximate value of $\mathrm{F_w}$. Then, let us note that there exists a positive value γ_{w} such that $(1 + \gamma_{\mathrm{w}})\overline{\mathrm{F}}_{\mathrm{w}}\tau \geq \mathrm{F_w}(\tau - 2\mathrm{f_w})$. We set the workload $\alpha_{\mathrm{w,0}}$ as follows:

$$\alpha_{\mathrm{w,0}} \equiv (1 + \gamma_{\mathrm{w}})\overline{\mathrm{F}}_{\mathrm{w}}\tau. \qquad (14)$$

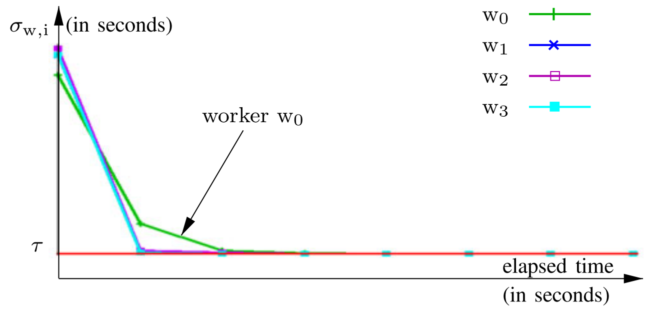


Fig. 7. Convergence of $\sigma_{w,i}$.

From an arbitrarily fixed value, γ_w can be iteratively increased until no contention be noticed at the very first round. According to the monotonic decrease of $\alpha_{w,i}$, we are sure, when such a value of γ_w has been found, that no contention will occur in the future.

Let's note that the larger the d_w time intervals, the lower the risk of contentions, in case of variation according to time of the execution parameters.

Besides, the time intervals d_w should allow all the workers to be served during the first round, i.e., within a τ period. Thus τ must verify:

$$\sum_{w=0}^{W-1} d_w \leq \tau. \tag{15}$$

In Section 6 we will consider a resource selection method to take into account constraint (15). In the mean time, in Sections 3, 4, and 5, we will assume that W, the number of workers, is small enough to make condition (15) hold.

Fig. 7 shows an example simulation result for four workers which illustrates Proposition 1. To get this illustration, we considered a star-shaped platform with a master connected to four workers $(w_i)_{i=0,3}$ by bidirectional links.

Each worker of this platform is characterized by one of the four profiles: $(R_i)_{0\leq i\leq 3}$, described in Table 1, which gives the value of the corresponding execution parameters. Latencies and speeds are given in seconds and MB/s respectively.

Each of the four curves (one per worker) represented on Fig. 7 illustrates the evolution of $\left(\sigma_{w,i}\right)_i$ as a function of time for the corresponding worker. As can be seen on Fig. 7, the value of $\left(\sigma_{w,i}\right)_i$ converges towards τ monotonically.

From the contents of Table 1, it can be claimed that the considered platform is heterogeneous. For instance, it takes worker w_0 6.0 seconds to process a chunk of unit size whereas it takes worker w_2 only 0.2 seconds. As a result, the duration to process a chunk of unit size can turn out to be about 30 times longer, from one worker to another one.

TABLE 1
Execution Parameter Values

	computation		communication master$\longrightarrow$w$_i$		communication master$\longleftarrow$w$_i$	
	speed	latency	speed	latency	speed	latency
R_0	0.08	3.00	0.90	0.80	0.90	2.00
R_1	0.08	0.10	0.93	0.10	0.93	0.80
R_2	0.05	0.10	0.96	0.90	0.96	1.00
R_3	0.06	0.10	0.91	0.60	0.91	0.90

TABLE 2
Values of ι_k

ι_0	ι_1	ι_2	ι_3	ι_4	ι_5
0.00	0.18	0.36	0.54	0.72	0.90

In order to illustrate the adaptivity of the AS4DR scheduler, the estimate of the computation speed of each worker (used by the method for the very first round of this simulation) has been willfully overestimated up to ten times the mean value. In spite of this deliberately bad estimation of one of the execution parameters, Fig. 7 shows that AS4DR adapts the chunksize so that the round duration for each worker converges rapidly towards τ. For this elementary example, $\sigma_{w,i}$ converges slower for worker w_0 than for the other workers because its computation latency is 30 times larger than that of the others (see Table 1).

3 Experimental Assessment of Adaptation to Unawareness in a Steady Context

In order to assess the relevance of AS4DR to adapting the workload when the execution parameters are poorly estimated, we want to compare it to another scheduler. As mentioned previously, no other scheduler maximizes the throughput when the communication model is one-port. So, we only compare AS4DR to a "Baseline scheduling" (BS) method, which is identical to AS4DR except that the workload is not adapted at each round; in other words assignment (1) is not applied.

With τ and an estimate of the available computation speed of worker w, denoted $\overline{F}_w$, it is possible, due to (14), to compute $\alpha_{w,0}$, an initial workload for each worker w. In order to be able to compare the two methods efficiently, the same value of τ is used for both methods: BS and AS4DR.

For the comparison of the simulation results, let us define CPU_{eff} the CPU-efficiency:

$$CPU_{eff} \equiv 1 - \frac{CPU_{idleness} + CPU_{latencies}}{\text{elapsed time}}; \tag{16}$$

where $CPU_{idleness}$ and $CPU_{latencies}$ respectively denote the time wasted in data-starvation and the time wasted in latencies. When the execution parameters are exactly known and steady, both schedulers make the workers process data without idleness, except time wasted in latencies.

For the simulation, we consider a platform with 1,000 workers and 10 sets of reference values $(R_k)_{0\leq k\leq 9}$ for the execution parameters of these workers, given in Table 3 (speeds in bytes/second and latencies in seconds). Each of these sets is randomly allocated to 100 workers among the 1,000 workers which constitute the platform.

In order to assess the effect of the inaccuracy of the execution parameters estimates on both schedules, the value of $\overline{F}_w$ is set by penalizing the reference value $(F_w)_{ref}$ given in Table 3:

$$\overline{F}_w := (1 \pm \iota_k)(F_w)_{ref}; \tag{17}$$

where ι_k is a positive real number which values are given in Table 2 and where the operator $\pm$ means that the operation performed is randomly chosen between either

TABLE 3
Execution Parameters Reference Values

	computation		communication master$\longrightarrow$w$_i$		communication master$\longleftarrow$w$_i$		number of
	speed	latency	speed	latency	speed	latency	workers
R_0	1.000	0.0100	10^6	0.010	10^6	0.010	100
R_1	100.0	0.1000	10^7	0.001	10^7	0.001	100
R_2	10.00	0.0010	10^8	0.015	10^8	0.015	100
R_3	1000.	0.0100	10^9	0.010	10^9	0.010	100
R_4	1.000	0.0010	10^6	0.010	10^6	0.010	100
R_5	100.0	0.1000	10^7	0.001	10^7	0.001	100
R_6	10.00	0.0001	10^8	0.010	10^8	0.010	100
R_7	1000.	0.0100	10^9	0.010	10^9	0.010	100
R_8	1.000	0.0010	10^6	0.001	10^6	0.001	100
R_9	100.0	0.0100	10^7	0.010	10^7	0.010	100

addition or subtraction. The values of ι_k characterize the inaccuracy of the execution parameters estimates; inaccuracy is minimum with ι_0, whereas it is maximum with ι_5. It is supposed to be the same for all the workers. Fig. 8 shows the measured CPU-efficiency of the whole platform for each scheduler, as a function of estimates' inaccuracy. For this simulation, which lasted 2,000 seconds, the value of τ has been arbitrarily set to 3 seconds. When the execution parameters are exactly known (ι_0), both schedulers offer the same CPU-efficiency: 99.99 percent. The computation latencies are responsible for the gap with the theoretical maximum CPU-efficiency: 100 percent. As expected, CPU-efficiency decreases for both methods when estimates inaccuracy increases. This decrease is considerably slower for AS4DR than for BS. For instance, when inaccuracy is maximum (ι_5), the CPU-efficiency with AS4DR is 1.89 times higher than with BS.

Figs. 9 and 10 show, for both schedulers and maximum estimates inaccuracy (ι_5), the variation (during the very first rounds of the streaming phase) of the CPU-efficiency of a particular worker. On both figures, the first scheduling point occurs beyond the end of the start-up phase. In abscissa are reported the dates of CPU-efficiency measurement: after the master received the result for the subchunk 1 of the first round, and after it received the result for the subchunk 2 of the other rounds. Beyond 26 seconds, the CPU-efficiency of this worker becomes steady, for both schedulers. With AS4DR (Fig. 10), the existence of an asymptotic limit to $\sigma_{w,i}$ explains the asymptotical character of the evolution of the CPU-efficiency. For both schedulers this asymptotic CPU-efficiency is numerically reached from the very first rounds. With AS4DR, the smallness of the computation latency of the worker under consideration: 10^{-4} seconds, in front of τ: 3 seconds, explains the magnitude of the speed of convergence; after just a few rounds, the CPU-efficiency with AS4DR for this worker is and stays 15.41 times higher than the one obtained with BS. It can also be noted that the dates of end of rounds (in abscissa) become asymptotically periodic for the worker under consideration. With AS4DR, the values $(\sigma_i)_{3\leq i\leq 9}$ are identical and equal τ: 3, whereas the values $(\sigma_i)_{2\leq i\leq 6}$ are equal to 4.62, with BS. The period of the asymptotical schedule installed with the BS method lasts approximately 4.62 seconds; which include lapses of idleness.

In order to estimate the duration $(\sigma_{w,i})_{i\geq 0}$ of a round for all the workers, Fig. 11 shows its average and its standard deviation over all the rounds and for all the workers, for the different values of estimates inaccuracy in Table 2.

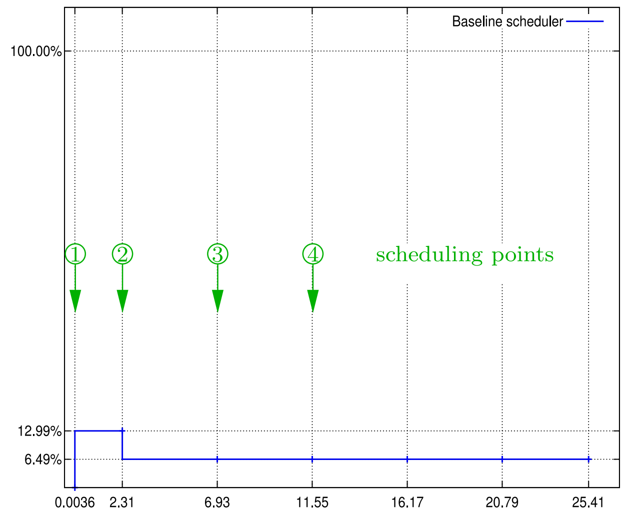


Fig. 9. BS: CPU-efficiency as a function of time (in seconds).

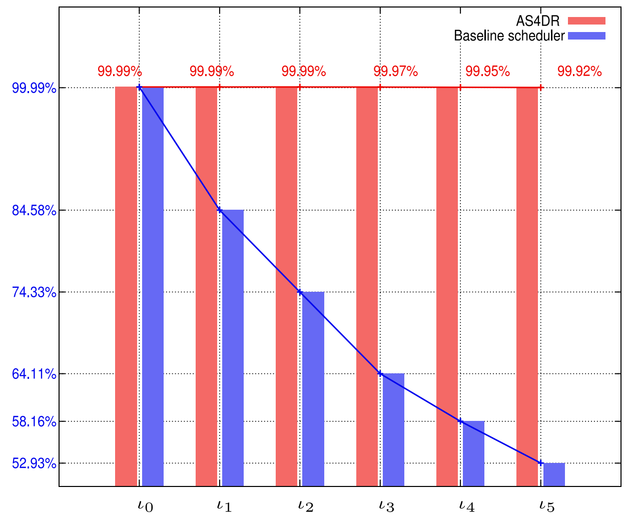


Fig. 8. CPU-efficiency as a function of inaccuracy.

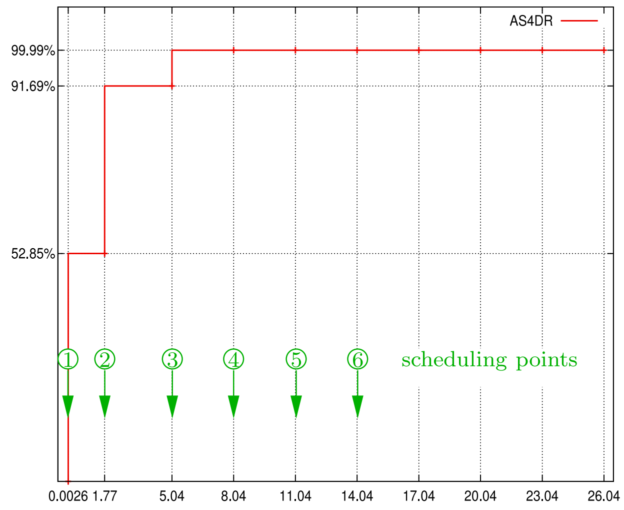


Fig. 10. AS4DR: CPU-efficiency as a function of time (in seconds).

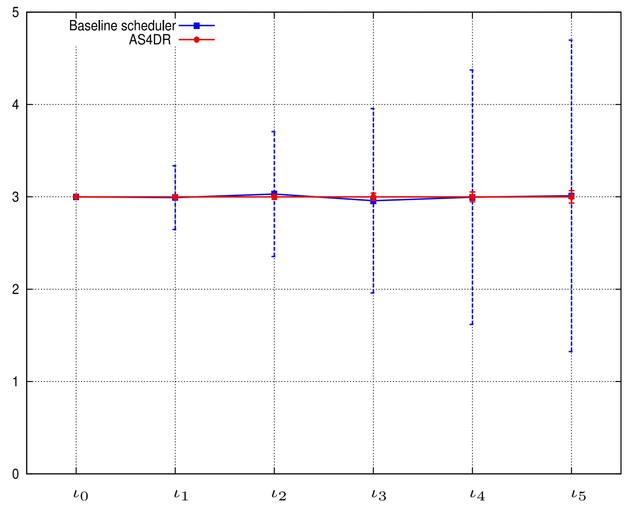


Fig. 11. Average and standard deviation of σ as a function of inaccuracy.

For both methods, Fig. 11 shows, on the one hand, that the average of the values of $(\sigma_{w,i})_{i\geq 0}$ is close to the desired value τ and, on the other hand, that the standard deviation of these values increases when estimates inaccuracy increases. It can be noticed that the standard deviation increases much faster with BS, in relation to estimates inaccuracy, than with AS4DR. For instance, the standard deviation with BS is 24.98 times higher than with AS4DR for ι_5. The ratio: 1.89, observed between the CPU-efficiency of both schedulers for ι_5 on Fig. 8, is a consequence of this disparity between the standard deviations of the values of $(\sigma_{w,i})_{i\geq 0}$. By adapting at each round the workload for each worker AS4DR reduces, on the one hand, the risk of contentions inherent to the one-port communication model and, on the other hand, the risk (in round-robin mode) of waiting for the result of a previous worker.

The next two sections assess to what extent AS4DR's ability to adapt the chunksize to the unawareness of execution parameters can be put to good use in adapting the chunksize to the variations of the values of these parameters over time.

4 Empirical Study of AS4DR in Dynamic Contexts

We consider a platform composed of a master node linked to four worker nodes $(w_k)_{0\leq k\leq 3}$. The values of the execution parameters considered for this platform are reference values $(R_k)_{0\leq k\leq 3}$ in Table 3.

In order to ease the interpretation of the simulation results, we consider basic variations of the execution parameters:

- only worker w_1 is concerned,
- only one of the speeds F_w, B_w^D or B_w^R varies at a time,
- an elementary variation of the value of a speed, called perturbation from now on, consists in a temporary reduction of the resource availability.

Among the characteristics of the fluctuations of the chosen execution parameter, we will successively consider the frequency, the amplitude and the smoothness of the perturbation. Let us denote T_{eff} the effective throughput:

$$T_{eff} \equiv \frac{\text{amount of processed data}}{\text{elapsed time}}.$$

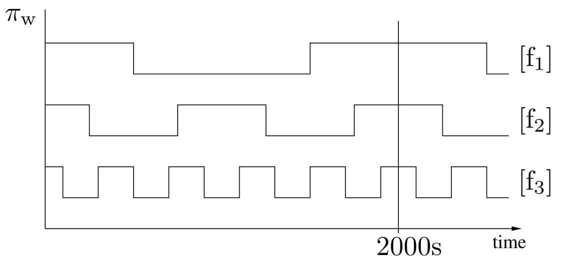


Fig. 12. Different perturbation frequencies.

4.1 Varying Perturbation Frequency

Let us first make the reductions of the resource availability more or less frequent. Precisely, we choose to have 1, 2 or 5 reductions of the availability during a 2,000 seconds time period and we denote f_1, f_2 and f_3 the respective corresponding dynamic contexts. They are illustrated in Fig. 12; where π_w is any one of the disturbed execution parameters among: F_w, B_w^D or B_w^R. During each fluctuation, the resource availability reduction for w_1 amounts to 30 percent of its nominal value. Figs. 13 and 14 respectively show that T_{eff} roughly follows the evolution of the computation speed F_1 for worker w_1 and remains almost constant for worker w_2.

Tables 4 and 5 respectively give the CPU_{eff} and the T_{eff} over the whole execution, for the different perturbation frequencies, and for each of the three considered execution parameters (F_w, B_w^D and B_w^R). As was very predictable, the global estimators CPU_{eff} and T_{eff} are decreasing functions of the frequency of the perturbation. Let us now explain the reason of this decrease. The number or rounds per worker being the same for all these simulations, the CPU latencies cannot be responsible for it. So the variation of both CPU_{eff} and T_{eff}, according to the frequency of the perturbation, comes necessarily from the idleness variation due to data-starvation.

To estimate the impact of the frequency of the perturbation on the idleness, Table 6 gives the ratio of the platform idleness over the total execution time, for the different perturbation frequencies, and for each of the three considered execution parameters. It shows that the considered

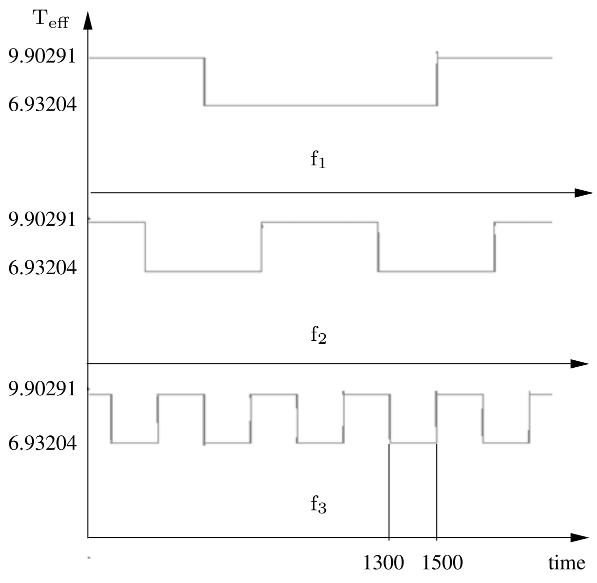


Fig. 13. T_{eff} of workers w_1 for $f*$-like computation speed.

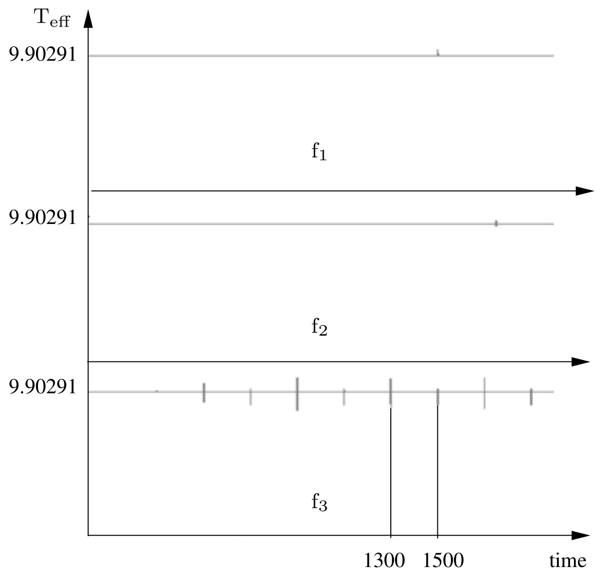


Fig. 14. T_{eff} of workers w_2, for $f*$-like computation speed.

TABLE 4
CPU_{eff} According to Perturbation Frequency

	CPU_{eff}		
	computation	communication master$\longrightarrow w_1$	communication master$\longleftarrow w_1$
f_1	99.1569 %	99.1569 %	99.1569 %
f_2	99.1554 %	99.1569 %	99.1569 %
f_3	99.1152 %	99.1569 %	99.1569 %

TABLE 5
T_{eff} According to Perturbation Frequency

	T_{eff}		
	computation	communication master$\longrightarrow w_1$	communication master$\longleftarrow w_1$
f_1	19.4119	20.9013	20.9013
f_2	19.4117	20.9013	20.9013
f_3	19.4094	20.9013	20.9013

TABLE 6
Idleness Ratio According to Perturbation Frequency

	Idleness duration / Total duration		
	computation	communication master$\longrightarrow w_1$	communication master$\longleftarrow w_1$
f_1	0. %	0. %	0. %
f_2	0.004 %	0. %	0. %
f_3	0.031 %	0. %	0. %

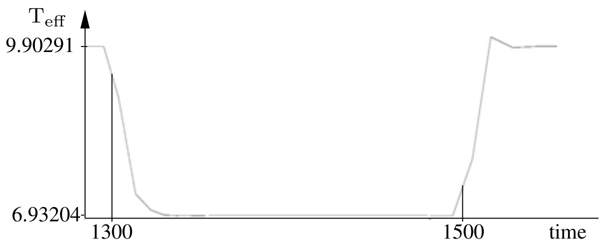


Fig. 15. T_{eff} of w_1 for f_5 in the range 1,300:1,500 s.

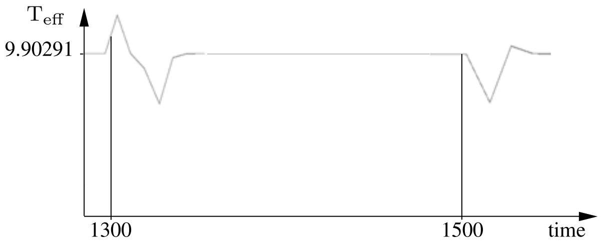


Fig. 16. T_{eff} of w_2 for f_5 in the range 1,300:1,500 s.

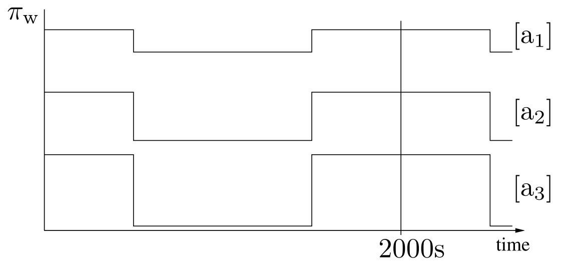


Fig. 17. Different perturbation amplitudes.

perturbations of the communication speed on the link between the master and worker w_1 are not sufficient to produce any worker idleness. That is, these perturbations do not prevent computation from overlapping communication. Table 6 also shows that the considered perturbation of the computation speed for worker w_1 becomes sufficient to introduce some idleness in contexts f_2 and f_3, when the computation speed varies quickly enough.

The starvations observed are inter-round ones, and affect workers w_0, w_2 and w_3 when reductions of the computation speed occur. The effects of the perturbation are absorbed in only one round.

In order to get a deeper understanding of the ability of the AS4DR method to adapt the scheduling to parameter variations, we now focus on computation speed fluctuations according to context f_3 over the time interval [1,300 s, 1,500 s], during which the computation speed F_1 of worker w_1 is reduced. Figs. 15 and 16 show the effect on T_{eff} of the delay in adapting to the computation speed variations which causes worker idleness, thus CPU_{eff} reduction. In order to investigate whether the value of θ_w is responsible for the inter-round idleness observed in context f_3, we artificially replaced this value by smaller ones: no significant reduction of idleness could be observed. This shows that Proposition 2 cannot be extended to a dynamic context without further hypothesis, in particular about dynamicity. The delayed adaptation of $\alpha_{w,i}$ to new values of the execution parameters by the scheduler is very likely responsible for the inter-round idleness observed.

4.2 Varying Perturbation Amplitude

We study the behaviour of AS4DR when the reductions of the resource availability have different amplitudes: 30, 60 and 95 percent, all other things being equal. We consider exactly one reduction per time period of 2,000 seconds. Fig. 17 illustrates the dynamic contexts a_1, a_2 and a_3 corresponding to these three types of perturbation.

TABLE 7
$\mathrm{CPU_{eff}}$ According to Perturbation Amplitude

	$\mathrm{CPU_{eff}}$		
	computation	communication master$\longrightarrow$$w_1$	communication master$\longleftarrow$$w_1$
a_1	99.1569 %	99.1569 %	99.1569 %
a_2	99.0526 %	99.1569 %	99.1569 %
a_3	82.1044 %	99.1569 %	99.1569 %

TABLE 8
$\mathrm{T_{eff}}$ According to Perturbation Amplitude

	$\mathrm{T_{eff}}$		
	computation	communication master$\longrightarrow$$w_1$	communication master$\longleftarrow$$w_1$
a_1	19.4119	20.9013	20.9013
a_2	17.8838	20.9013	20.9013
a_3	14.5244	20.9013	20.9013

TABLE 9
Idleness Ratio According to Perturbation Amplitude

	Idleness duration / Total duration		
	computation	communication master$\longrightarrow$$w_1$	communication master$\longleftarrow$$w_1$
a_1	0. %	0. %	0. %
a_2	0.034 %	0. %	0. %
a_3	11.628 %	0. %	0. %

Tables 7 and 8 respectively give the $\mathrm{CPU_{eff}}$ and the $\mathrm{T_{eff}}$ over the total execution, for different perturbation amplitudes and for each of the three considered execution parameters. The global estimators $\mathrm{CPU_{eff}}$ and $\mathrm{T_{eff}}$ are decreasing functions of the amplitude of the perturbation.

Table 9 shows that only the considered perturbation of the computation speed for worker w_1 becomes sufficient to introduce some inter-round idleness, when the amplitude is large enough as in a_2 or a_3.

The amplitude values a_1, a_2 and a_3 have been chosen so that the reduction of the availability of the considered resource (computation or communication), on average over the range 0:2,000 seconds, be respectively the same as for the frequency values f_1, f_2 et f_3. This choice makes possible the comparison between the effects of various types of perturbations, on the efficiency. When average variations are similar, it seems that the variations of the perturbation in amplitude affect the worker's idleness more than variations of the perturbation in frequency. This suggests that the greater the smoothness of the variation, the smaller the loss of efficiency. Next section strengthens this intuition on another example.

4.3 Varying Perturbation Smoothness

Finally, in order to assess the effect of the perturbation smoothness on the scheduling adaptation, we now consider the AS4DR method in a context φ_3 similar to f_3, but with smoother speed variations (cf Fig. 18). Precisely, every decrease or increase made in one step in context f_3 is decomposed in four successive steps in context φ_3.

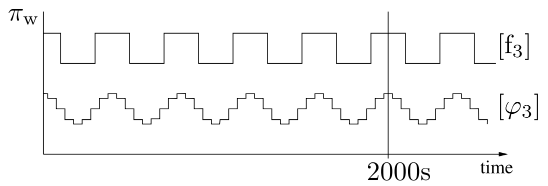


Fig. 18. Different perturbation smoothnesses.

TABLE 10
Relative Idleness According to Perturbation Smoothness

	Idleness duration / Total duration		
	computation	communication master$\longrightarrow$$w_1$	communication master$\longleftarrow$$w_1$
f_3	0.031 %	0. %	0. %
φ_3	0. %	0. %	0. %

Table 10 confirms that, for similar resource variations, the idleness is lower when the variation is smoother (no idleness at all in this particular case).

For these comparable frequency and amplitude of the perturbation, the slower variation of the computation speed F_1 of w_1, in context φ_3 compared to context f_3, allows the AS4DR scheduling to suffer less from the delay of adaptation.

5 Experimental Assessment of Adaptation to Dynamicity

In this section, the reference values for the execution parameters, given in Table 3, are randomly allocated to 1,000 workers, as previously. But, these values are likely to vary over time, according to the 10 profiles $(P_k)_{0\leq k\leq 9}$ of variation shown in Fig. 19. Whatever the profile P_k being allocated to a worker w and whatever the execution parameter, the higher value of P_k equals the reference value allocated to the worker w, for the execution parameter under consideration; the lower value of P_k is computed from the reference value as:

$$B_w^D := (1-\delta)\big(B_w^D\big)_{ref}, B_w^R := (1-\delta)\big(B_w^R\big)_{ref}, \quad (18)$$

$$F_w := (1-\delta)(F_w)_{ref}; \quad (19)$$

where δ is a positive parameter, smaller than one, the values of which are given in Table 11.

In this context, the coefficient δ characterizes the variation of the execution parameters and is called "dynamicity" in the sequel; when δ equals δ_4, the variations amplitude is maximum, whereas it is minimum when δ equals δ_0. For each simulation, the dynamicity is the same for all the workers. Whatever the considered worker, the perturbations of its different speeds are based on the same profile. The profiles $(P_k)_{0\leq k\leq 9}$ are randomly allocated to the 1,000 workers. Fig. 20 shows the measured CPU-efficiency of the whole platform for each scheduler, as a function of the dynamicity. For this simulation, which lasted 2,000 seconds, the value of τ has been arbitrarily set to 3 seconds. When the execution

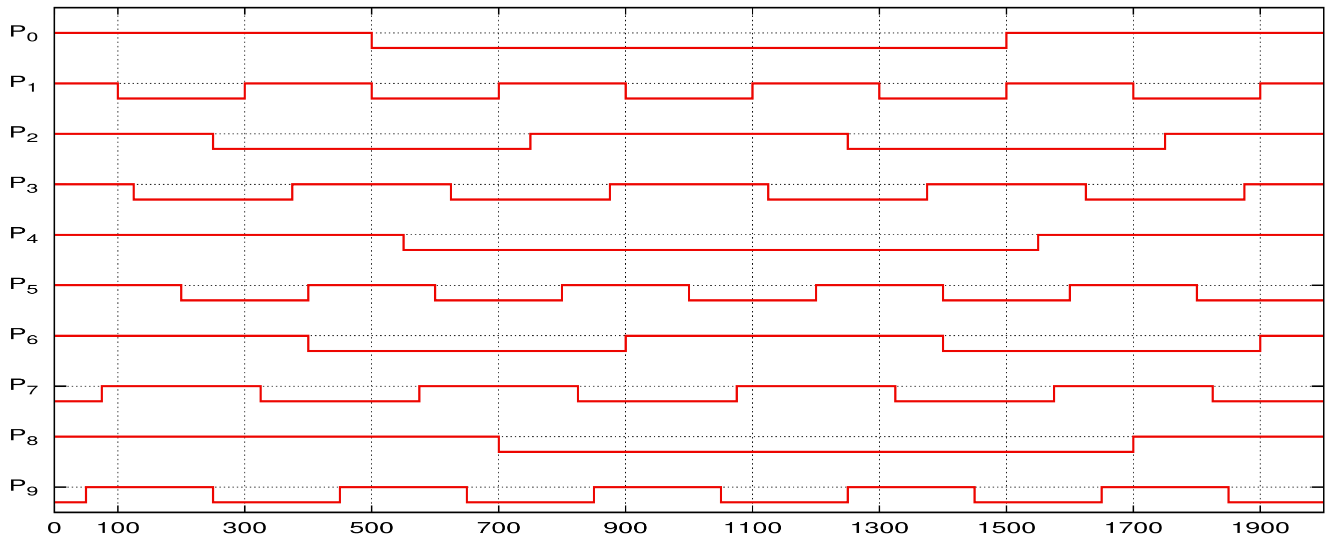


Fig. 19. Disturbed execution parameter as a function of time (in seconds).

parameters are steady, that is when δ equals δ_0, both schedulers offered the same CPU-efficiency: 99.99 percent. As expected, the CPU-efficiency for both methods decreases when the dynamicity of the execution parameters increases. This decrease is significantly faster for BS than for AS4DR. For instance, when the dynamicity equals δ_4, the CPU-efficiency with AS4DR is 1.38 times higher than with BS.

Figs. 22 and 23 show, for both schedulers, the variation of the CPU-efficiency of a worker with profile P_1 during the first reduction of the value of the execution parameters, when the dyna; that is when δ equals δ_4. After 100 seconds, each execution parameter of the worker under consideration is reduced by 80 percent. Then, 200 seconds later, each execution parameter recovers its reference value. With AS4DR (Fig. 23), the smallness of the computation latency of the worker, in front of τ, explains the quickness (three rounds) of the adaptation to the change of context, on the falling edge as well as the rising one. In the absence of adaptation to the change of the characteristics of the platform, the CPU-efficiency observed with BS (Fig. 22) is reduced for a long time (68 rounds).

Fig. 21 shows the average and the standard deviation of $(\sigma_{\mathrm{w,i}})_{\mathrm{i}\geq 1}$ over all the rounds and for all the workers, for different values of the dynamicity, when considering the profile P_1. It can be noted that, with BS, the average of the values $(\sigma_{\mathrm{w,i}})_{\mathrm{i}\geq 1}$ is an increasing function of dynamicity. The reason for this is that, due to (18) and (19), the perturbation is a reduction of the reference value of the execution parameters, given in Table 3; which systematically increases the duration of the rounds. On the contrary, with AS4DR, the average of the values $(\sigma_{\mathrm{w,i}})_{\mathrm{i}\geq 1}$ stays close to the desired value τ. For both methods, Fig. 21 shows that the standard deviation of these values is an increasing function of dynamicity. Once again, the better CPU-efficiency observed with AS4DR compared to that observed with BS, for instance for δ_4 on Fig. 20, is a consequence of the disparity between the standard deviations of the values of $(\sigma_{\mathrm{w,i}})_{\mathrm{i}\geq 1}$, shown by Fig. 21.

TABLE 11
Values of δ

δ_0	δ_1	δ_2	δ_3	δ_4
0.0	0.2	0.4	0.6	0.8

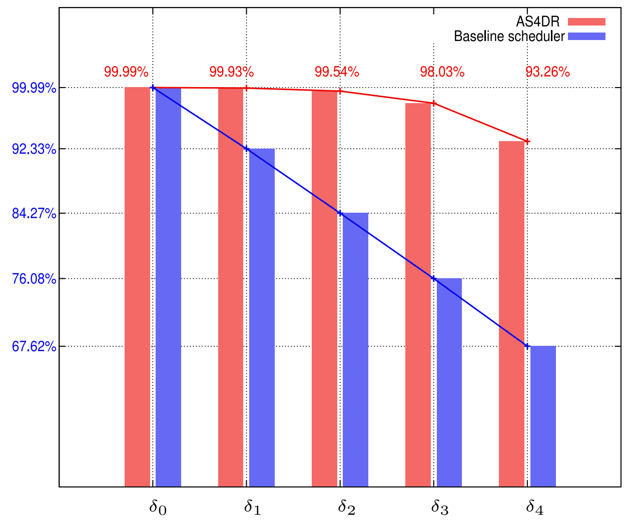


Fig. 20. CPU-efficiency as a function of dynamicity.

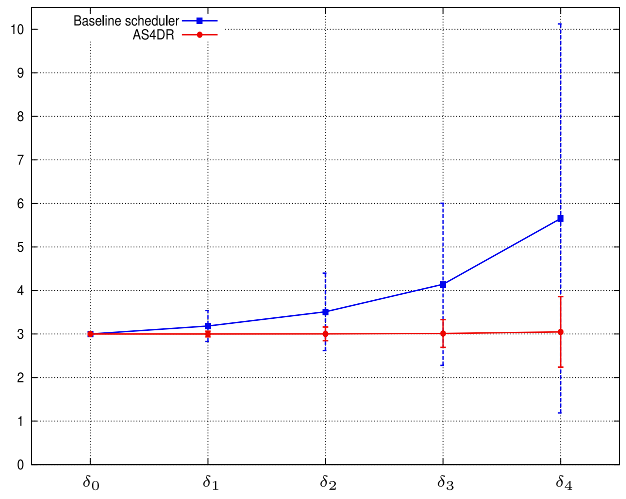


Fig. 21. Average and standard deviation of σ as a function of dynamicity.

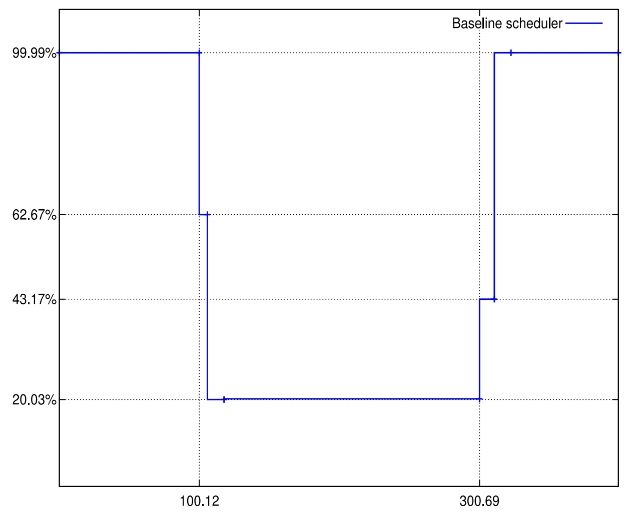


Fig. 22. BS: CPU-efficiencyas a function of time.

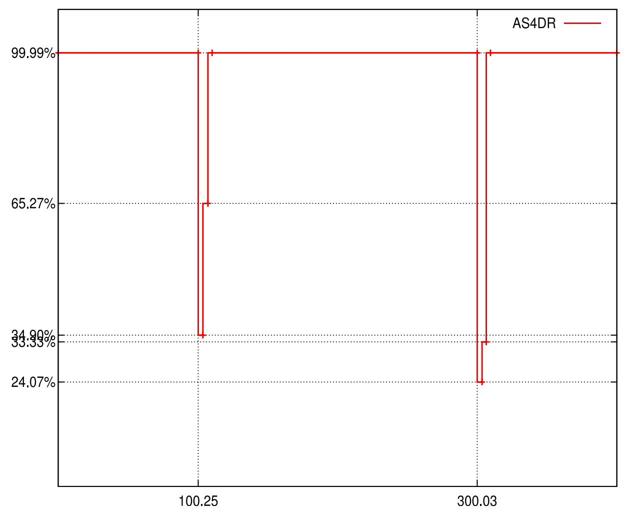


Fig. 23. AS4DR: CPU-efficiency as a function of time.

6 Resource Selection for AS4DR in a Multiple Data Streams Context

When implementing the master-workers paradigm, the well-known drawback is the non-scalability of the model, for the number of workers allocated to one master cannot endlessly increase without contentions occurring, as workers compete to access the master. When scheduling with AS4DR, the requirement that the number of workers allocated to one master be small enough, has led to inequality (15), for each master-workers cluster. This section presents a resource selection method which aims to set up an upper bounded number of master-workers clusters, enabling the maximization of the throughput of the processed data.

6.1 Method

Let $\mathcal{W}$ and M be respectively the set of all available workers and the upper bound of the number of star-shaped platforms to set up. Let $\mathrm{x_{m,w}}$ equals one if the worker w is selected in the cluster m, and $\mathrm{x_{m,w}}$ equals zero otherwise. As one worker can belong to one cluster at most,

$$\sum_{\mathrm{m=0}}^{\mathrm{M-1}} \mathrm{x_{m,w}} \leq 1, \quad \forall \mathrm{w} \in \mathcal{W}. \tag{20}$$

Let T be the global throughput,

$$\mathrm{T} = \sum_{\mathrm{m=0}}^{\mathrm{M-1}} \sum_{\mathrm{w}\in\mathcal{W}} \mathrm{t_w x_{m,w}}; \tag{21}$$

where $\mathrm{t_w}$, the potential throughput of worker w in the absence of data-starvation between the processing of any successive subchunks, verifies: $\mathrm{t_w} = (1 - 2\frac{\mathrm{f_w}}{\tau_\mathrm{m}})\mathrm{F_w}$; where τ_m is the desired duration for the rounds of cluster m, which worker w belongs to. In order to prevent contentions, and thus to make the use of AS4DR possible, we have seen (15) that necessarily,

$$\sum_{\mathrm{w}\in\mathcal{W}} \mathrm{d_w x_{m,w}} \leq \tau_\mathrm{m}, \quad \forall\, 0 \leq \mathrm{m} \leq \mathrm{M}-1. \tag{22}$$

To form a set of clusters which maximizes the global throughput, when considering M data streams at most, we want to find $(\mathrm{x_{m,w}})_{0\leq \mathrm{m}\leq \mathrm{M-1}, \mathrm{w}\in\mathcal{W}}$ that maximizes T, given by (21), subject to constraints (22) and (20). This problem looks like a Multiple Knapsack problem. But as the value of τ_m cannot be set a priori, since it depends on the workers selected for cluster m, the same desired duration τ is set for the rounds of all clusters. This choice has the advantage of upper bounding the load discrepancy between the different clusters uniformly.

Besides, the time lag $\mathrm{d_w}$, defined by (13), depends on the worker w-1, the one that precedes w in the round robin distribution for the cluster which w belongs to. As this worker w-1 is a priori unknown, we need to reformulate the Multiple Knapsack problem to make the weight for worker w be independent from worker w-1. For that, let us tighten up the constraints of the problem. We define $\overset{\circ}{\mathrm{d}}_\mathrm{w}$, as follows:

$$\overset{\circ}{\mathrm{d}}_\mathrm{w} \equiv \max\Big(\overline{D_{\max}}, \overline{\ddot{R}_{\max}} + \overline{\dot{R}_{\mathrm{w},0}}\Big);$$

$$\text{where } \begin{cases} \overline{\mathrm{D}_{\max}} & \equiv \max_{\mathrm{w}\in\mathcal{W}}\Big(\overline{\dot{D}_{\mathrm{w},0}} + \overline{\ddot{D}_{\mathrm{w},0}}\Big), \\ \overline{\ddot{\mathrm{R}}_{\max}} & \equiv \max_{\mathrm{w}\in\mathcal{W}} \overline{\ddot{\mathrm{R}}_{w,0}}. \end{cases}$$

As $\mathrm{d_w}$ is smaller than $\overset{\circ}{\mathrm{d}}_\mathrm{w}$, the feasible space defined by the following constraints:

$$\sum_{\mathrm{w}\in\mathcal{W}} \overset{\circ}{\mathrm{d}}_\mathrm{w} \mathrm{x_{m,w}} \leq \tau, \quad \forall\, 0 \leq \mathrm{m} \leq \mathrm{M}-1, \tag{23}$$

is a subspace of the one previously defined by (22). So, the tightened problem we have to solve now is: find $(\mathrm{x_{m,w}})_{0\leq \mathrm{m}\leq \mathrm{M-1}, \mathrm{w}\in\mathcal{W}}$ that maximizes T, given by (21), subject to constraints (23) and (20).

Unfortunately, the coefficients $\mathrm{t_w}$, $\overset{\circ}{\mathrm{d}}_\mathrm{w}$ and τ are not positive integers, as in the classical Multiple Knapsack problem. According to the machine number representation, they are rational numbers. So, in order to make the problem be formulated like a Multiple Knapsack problem, (21) can be multiplied by the least common multiple of the denominators of $(\mathrm{t_w})_{w\in\mathcal{W}}$ whereas, (23) can be multiplied by the least

TABLE 12
Values of δ

δ_0	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9
0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

common multiple of the denominators of $(\mathrm{d_w})_{w\in\mathcal{W}}$ and τ. This new problem can be solved in a classical way by a Multiple Knapsack method. If the whole set of workers happens to be selected to participate in the processing, then the solution of this problem maximizes the global throughput, for a given desired duration for the rounds τ. Of course, as the presented method to select the workers is based on the underlying scheduling method, AS4DR, it is not possible to claim that the global throughput is optimal if only a part of the whole set of workers happens to be selected. Obviously, there exists a value of M big enough to make all the workers be ultimately involved in the schedule. Because of the successive transformations of the resource selection problem, the number of actually formed clusters is possibly no more minimal (but still smaller than M).

The next section is devoted to the experimental assessment of the resource selection method. The method described in [23] has been chosen to solve the Multiple Knapsack problem posed by the resource selection.

6.2 Experimental Assessment

We still allocate the reference values given in Table 3 to 1,000 workers randomly, as previously. To make $\mathrm{F_w}$, $\mathrm{B_w^D}$ and $\mathrm{B_w^R}$ vary over time, in a way that facilitates the correlation of the variations with their effects on the scheduling, each of these parameters can only take two values. According to the 10 profiles $(\mathrm{P_k})_{0\leq k\leq 9}$ of variation shown by Fig. 19, $(\mathrm{F_w}, \mathrm{B_w^D}, \mathrm{B_w^R})$ alternatively takes the reference value given by Table 3: $((\mathrm{F_w})_{\mathrm{ref}}, (\mathrm{B_w^D})_{\mathrm{ref}}, (\mathrm{B_w^R})_{\mathrm{ref}})$, and the disturbed value: $((1-\delta)(\mathrm{F_w})_{\mathrm{ref}}, (1-\delta)(\mathrm{B_w^D})_{\mathrm{ref}}, (1-\delta)(\mathrm{B_w^R})_{\mathrm{ref}})$; where the dynamicity δ takes the values given in Table 12.

Each profile is randomly allocated to 100 workers. With δ_9 the amplitude of the variation of the execution parameters is maximum, whereas it is minimum (steady context) with δ_0. For each simulation, the dynamicity is the same for all the workers.

Fig. 24 shows the variation of the global throughput (in bytes/second) as a function of M, the upper bound of the number of data streams. For each value of M, the total number of selected workers is given. The execution parameters for this simulation are those of Table 3; the dynamicity equals δ_0, and the a priori estimate of the execution parameters equals their real values. As best workers are selected in priority, the greater the number of workers involved in the scheduling, the lower the increase of the throughput.

Fig. 25 represents the mean value of the measured durations (in seconds) of the rounds: σ, as a function of the elapsed time (in seconds), when M is big enough for the whole set of 1,000 workers to be selected. For this experiment, τ equals 100 seconds and the execution parameters do not vary over time ($\delta = \delta_0$). In order to illustrate the adaptivity of the scheduling to the poorness of the estimates of the execution parameters, the initial workload is set to −80 percent of the load computed with the aid of the real execution parameters; those of Table 3, whereas the latency of computation, for all the workers, has been set to: 0.1, 0.5 and 0.9, successively. Fig. 25 shows the convergence towards τ of the mean value of σ, thus the adaptation of the workload to the poorness of the estimation of the execution parameters, when the resources are automatically selected. As expected, the smaller the processing latencies, the faster the adaptation of σ to τ.

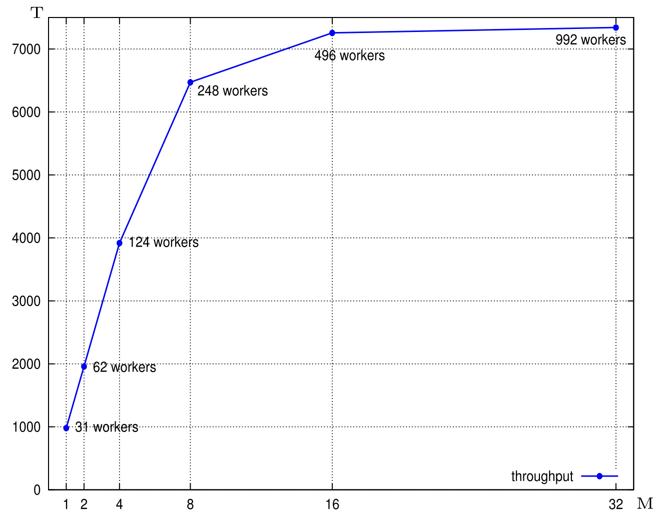


Fig. 24. $\mathrm{T_{eff}}$ as a function of the upper bound M.

Although the throughput remains the ultimate performance indicator, it does not highlight the rate of time really spent in processing like the CPU-efficiency does. This remark leads us to consider now the CPU-efficiency indicator, defined by (16).

Fig. 26 illustrates that the value of τ should depend on the computation latency; as long as these latencies are not negligible compared to τ.

This simulation has been performed in steady context ($\delta = \delta_0$), with the full knowledge of the execution parameters of Table 3, but the computation latencies for all the workers are replaced by the values: 0.0, 0.01 and 0.1, successively. Fig. 26 also recalls that, according to the value of the computation latency, there exists an optimal value for τ.

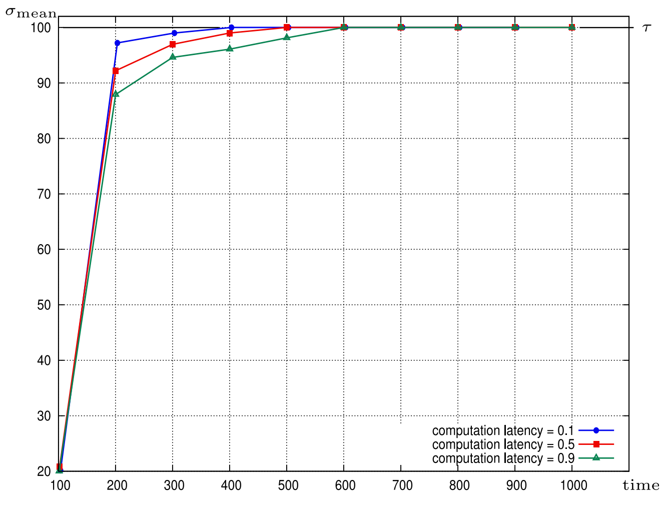


Fig. 25. Mean value of σ as a function of elapsed time.

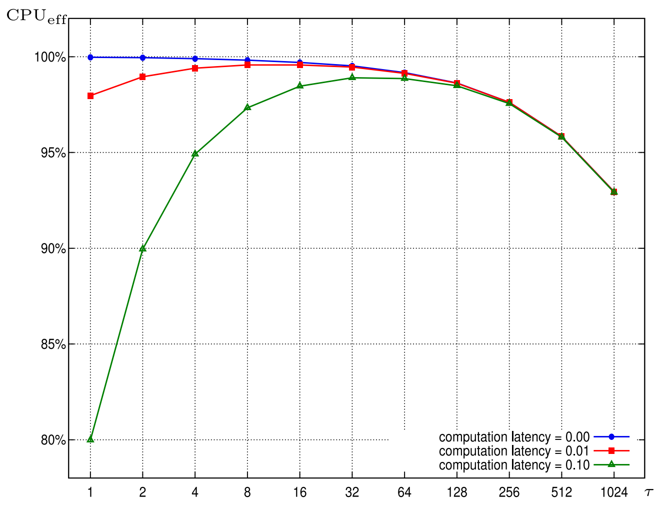


Fig. 26. $\mathrm{CPU_{eff}}$ as a function of τ, when dynamicity = δ_0.

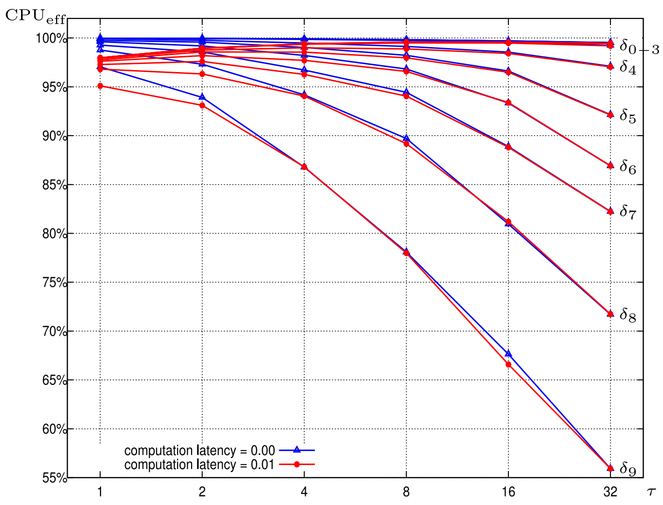


Fig. 27. $\mathrm{CPU_{eff}}$ as a function of τ for several dynamicity values.

The existence of such an optimal value is due to a compromise between two necessities: make τ as great as possible in order to minimize the time wasted in latencies (by reducing the number of rounds), and keep it small for the workers to start processing early.

Fig. 27 shows that the higher the dynamicity, the smaller τ must be. It also confirms that, even in a dynamic context, the smaller τ, the higher the effects on the efficiency of the time wasted in latencies. This simulation has been performed with several values of dynamicity, with the execution parameters of Table 3, but the CPU latencies for all the workers were replaced by the values: 0.0 and 0.01, successively.

7 Related Works

Although most general research work on scheduling addresses applications with intertask dependencies, quite a large number of applications [24] are eligible to the divisible load scheduling (DLS) model.

This model has been largely studied; it is the first model that enables optimality proofs for scheduling methods in the context we have chosen [3], [10], [25]. It usually deals with problems for which the size of the total workload and the execution parameters are known [26], [27].

Several multi-round methods have been proposed [13], [26], [27]. On the one hand the iterated distributions of multi-round methods have the advantage (when using a one-port model) of making the workers active earlier, and on the other hand they have the drawback of increasing the amount of time wasted by latencies. Several strategies for distributing the load to workers have already been studied. First of all, some strategies fix the number of rounds arbitrarily (multi-installment methods [13]). They are well suited to homogeneous platforms with a linear costs model. For heterogeneous platforms [1], other strategies have been proposed, which are able to take account of the non-linearity [28] of costs when determining the load of the nodes for each round [29]. For instance, the workload which is delivered at each round by the UMR method [27] 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 totally. It is proved [27] that this method minimizes the total execution time; provided that we know how to select the best set of nodes to be used. Another method [26] introduces periodicity for the rounds (without imposing any particular value for the period) and also requires that all nodes work during the whole period and that computation overlaps communication totally. It is proved [26] that it maximizes the amount of load processed by time unit.

Unfortunately none of these methods can be used when the execution parameters are inaccurately specified. To deal with such a lack of information, some methods probe to get the missing information in order to adjust the scheduling [15], [16], [30], [31], [32]. For other methods, like RUMR method [11], the law of probability which governs the fluctuation of the value of the execution parameters is supposed to be known. The problem of predicting the robustness of DLS scheduling in an environment characterized by unpredictable variations is addressed in [33]

The DA1 [15], OLMR [16] and AS4DR [17] methods can cope with non-specification of the total workload. They make the worker's access to the master become periodic in order to maximize the productive utilization of each of them. More precisely, from a contention-free situation, these methods can keep up an organisation of the successive rounds, which gives each worker dedicated time frames to dialog with the master. Each method improves the CPU-efficiency over the previous one: whereas the OLMR method aims at avoiding, in an assumed contention-free context, the workers idleness experienced by DA1's scheduling, by allowing computation to overlap communication, AS4DR avoids the data-starvation of the workers by preventing contentions inherent to the one-port communication model.

8 Conclusion

The AS4DR method presented in this paper succeeds in supplying, without starvation, data to processing units, when scheduling a divisible load of unknown total size on distributed resources with inaccurately specified, but steady, characteristics. Despite the fact that a one-port communication model is prone to contention, AS4DR can avoid this pitfall. To ease the contention prevention, a desired duration τ for all the rounds and for all workers is set a priori. Thanks both to the asymptotic periodicity of communication it installs (for both data and results) and to the delays, introduced at each round, between data communications, AS4DR prevents workers from data starvation. From any value of τ, an empirical method is offered to set the value of these delays before the very first round. For any given value of τ, the schedule produced with AS4DR maximizes the throughput, when considering an affine cost for computation; when considering a linear cost for computation, the optimal throughput value does not depend on the value of τ. Moreover, it has been proved that, when costs are affine for both computation and communication, the scheduling is stable. Some simulations illustrate the adaptation of the scheduling to the lack of awareness of the characteristics of the platform.

When the characteristics of the platform vary in an unpredictable way, there is no theoretical result. By means of simulations, we empirically studied the impact, on the CPU-efficiency and the throughput, of different aspects of the unpredictable variations of the execution parameters value: frequency, amplitude and smoothness. This study has shown that there exists a threshold for dynamicity under which AS4DR is able to provide a starvation-free scheduling when the value of execution parameters evolves over time. To assess the efficiency of the scheduler more thoroughly, when the characteristics of the platform vary over time, the method must be applied to real-world stream processing problems on a physical environment. In this context, the way the workload is adapted at each round generates a risk of instability. A study of the stability of the method should lead to tighten up the way τ is set and should suggest the design of new patterns of adaptation of the chunksize.

Of course, for a given set of execution parameters values and a given value of τ, using all the available resources will ultimately become impossible with an evermore increasing amount of resources. So, this paper also addresses the problem of setting up clusters of heterogeneous distributed resources to process several data streams with the AS4DR scheduler. For the moment the value of τ can only be empirically set, according both to the poorness of the a priori estimate of the execution parameters and to the frequency of their possible variation over time. If only a part of the whole set of available computation units are involved in the scheduling, then we can only claim that the selected computation units are fully used. But, if the whole set of available processing units are involved, then the presented method succeeds in maximizing the global throughput, for a given value of τ.

Compared to using pure hardware performance figures, such as bandwidth or CPU frequency to adapt the workload at each round, the method has the extra advantage of taking into account characteristics of the software such as algorithmic complexity.

References

[1] A. L. Rosenberg and R. C. Chiang, "Toward understanding heterogeneity in computing," in *Proc. 24th Int. Parallel Distrib. Process. Symp.*, Apr. 2010, vol. 1, no. 1, pp. 1–10.

[2] O. Beaumont, L. Marchal, and Y. Robert, "Scheduling divisible loads with return messages on heterogeneous master-worker platforms," *Lecture Notes in Comput. Sci.*, vol. 3769, pp. 498–507, 2005.

[3] T. G. Robertazzi, "Ten reasons to use divisible load theory," *IEEE Comput.*, vol. 36, no. 5, pp. 63–68, May 2003.

[4] C. Lee and M. Hamdi, "Parallel image processing application in a network of workstation," *Parallel Comput.*, vol. 21, pp. 137–160, 1995.

[5] D. Altilar and Y. Paker, "An optimal scheduling algorithm for stream based parallel video processing," in *Proc. 18th Int. Symp. Comput. Inf. Sci.*, 2003, pp. 731–738.

[6] L. Dong, V. Bharadwaj, and C. C. Ko, "Efficient movie retrieval strategies for movie-on-demand multimedia services on distributed networks," *Multimedia Tools Appl.*, vol. 20, no. 2, pp. 99–133, 2003.

[7] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang, "Scheduling divisible loads on star and tree networks: Results and open problems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 3, pp. 207–218, Mar. 2005.

[8] M. Drozdowski and P. Wolniewicz, "Optimizing divisible load scheduling on heterogeneous stars with limited memory," *Eur. J. Oper. Res.*, vol. 172, no. 2, pp. 545–559, 2006.

[9] T. Saif and M. Parashar, "Understanding the behavior and performance of non-blocking communications in MPI," in *Proc. 9th Int. Euro-Par Conf.*, 2004, pp. 173–182.

[10] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Computing Society Press, 1996.

[11] Y. Yang and H. Casanova, "Rumr: Robust scheduling for divisible workloads," in *Proc. 12th IEEE Int. Symp. High Perform. Distrib. Comput.*, 2003, pp. 114–125.

[12] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 4, pp. 319–330, Apr. 2004.

[13] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Trans. Aerospace Electron. Syst.*, vol. 31, no. 2, pp. 555–567, Apr. 1995.

[14] A. Legrand, Y. Yang, and H. Casanova, "Np-completeness of the divisible load scheduling problem on heterogeneous star platforms with affine costs," UCSD/CSE, Dept. Comput. Sci. Eng., Univ. California, San Diego, Tech. Rep. CS2005-0818, Mar. 2005.

[15] M. Drozdowski, "Selected problems of scheduling tasks in multiprocessor computing systems," Ph.D. dissertation, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.

[16] S. Boutammine, D. Millot, and C. Parrot, "Dynamically scheduling divisible load for grid computing," in *Proc. 2nd Int. Conf. High Perform. Comput. Commun.*, 2006, pp. 763–772.

[17] D. Millot and C. Parrot, "Scheduling on unspecified heterogeneous distributed resources," in *Proc. 25th Int. Symp. Parallel Distrib. Process. Workshops*, May 2011, vol. 0, no. 1, pp. 45–56.

[18] D. Millot and C. Parrot, "Some tests of adaptivity for the AS4DR scheduler," in *Proc. 41th Int. Conf. Parallel Process.*, Los Alamitos, CA, USA, Sep. 2012, pp. 323–331.

[19] D. Millot and C. Parrot, "First experimental assessments of the adaptivity of the scheduling with as4dr," in *Proc. 13th Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Dec. 2012, pp. 487–492.

[20] D. Millot and C. Parrot, "Setting up clusters of computing units to process several data streams efficiently," in *Proc. 10th Int. Conf. Parallel Process. Appl. Math.*, Sep. 2013, pp. 49–61.

[21] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014.

[22] V. Bharadwaj and G. Barlas, "Efficient scheduling strategies for processing multiple divisible loads on bus networks," *J. Parallel Distrib. Comput.*, vol. 62, no. 1, pp. 132–151, Jan. 2002.

[23] D. Pisinger, "An exact algorithm for large multiple knapsack problems," *Eur. J. Oper. Res.*, vol. 114, no. 3, pp. 528–541, 1999.

[24] A. Shokripour and M. Othman, "Categorizing DLT researches and its applications," *Eur. J. Sci. Res.*, vol. 37, no. 3, pp. 496–515, 2009.

[25] J. Sohn, T. G. Robertazzi, and S. Luryi, "Optimizing computing costs using divisible load analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 3, pp. 225–234, Mar. 1998.

[26] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," INRIA, Le Chesnay, France, Tech. Rep. RR-4595, Oct. 2002.

[27] Y. Yang and H. Casanova, "UMR: A multi-round algorithm for scheduling divisible workloads," in *Proc. Int. Parallel Distrib. Process. Symp.*, Apr. 2003, p. 24.

[28] R. W. Hockney, "The communication challenge for MPP: Intel paragon and meiko cs-2," *Parallel Comput.*, vol. 20, no. 3, pp. 389–398, 1994.

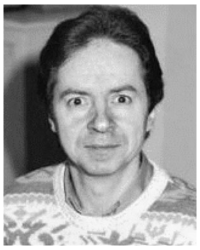
[29] J. T. Hung and T. G. Robertazzi, "Scheduling nonlinear computational loads," Stony Brook Univ., New York, NY, USA, Tech. Rep. CEAS 823, Sep. 2006.

[30] J. Jia, B. Veeravalli, and D. Ghose, "Adaptive load distribution strategies for divisible load processing on resource unaware multilevel tree networks," *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 999–1005, Jul. 2007.

[31] H. Li, G. Sun, and Y. Xu, "Distributed scheduling strategies for processing multiple divisible loads with unknown network resources," in *Proc. Int. Conf. Netw. Parallel Comput. Workshops*, 2007, pp. 824–829.

[32] D. Luong, J. Deogun, and S. Goddard, "Feedback scheduling of real-time divisible loads in clusters," *SIGBED Rev.*, vol. 5, no. 2, pp. 1–4, 2008.

[33] M. Balasubramaniam, I. Banicescu, and F. M. Ciorba, "Analyzing the robustness of scheduling algorithms using divisible load theory on heterogeneous systems," in *Proc. IEEE 12th Int. Symp. Parallel Distrib. Comput.*, 2013, pp. 45–52.



Daniel Millot received the PhD degree in computer science in 1991, from the Paris XI University. He is an associate professor at Telecom SudParis. His research interests include high performance computing: parallel algorithms, load balancing, and heterogeneous programming.

Christian Parrot received the PhD degrees in mathematics from the Université Paris 6, France, in 1993. He is currently an associate professor in the Computer Science Department of Telecom sudParis. His main research interest includes scheduling methods for distributed resources. He is a member of the IEEE.