Toward a Fully Decentralized Algorithm for Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin, Arnaud Legrand, Corinne Touati

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Toward a Fully Decentralized Algorithm for Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin — Arnaud Legrand — Corinne Touati

N° 6537

April 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6537--FR+ENG

Toward a Fully Decentralized Algorithm for Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin , Arnaud Legrand , Corinne Touati

Thème NUM — Systèmes numériques
Projet MESCAL

Rapport de recherche n° 6537 — April 2008 — 24 pages

Abstract: In this paper, we present a fully decentralized algorithm for fair resource sharing between multiple bag-of-tasks applications in a grid environment. This algorithm is inspired from related work on multi-path routing in communication network. An interesting feature of this algorithm is that it allows the choice of wide variety of fairness criteria and achieves both optimal path selection and flow control. In addition, this algorithm only requires local information at each slave computing tasks and at each buffer of the network links while minimal computation is done by the schedulers. A naive adaptation is unstable and inefficient though. Fortunately, a simple and effective scaling mechanism is sufficient to circumvent this issue. This scaling mechanism is motivated by a careful study of the subtle differences with the classical multi-path routing problem. We prove its efficiency through a detailed analysis of a simple simulation.

Key-words: Scheduling, distributed computing, lagrangian optimization

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

Vers un algorithme totalement décentralisé d'ordonnancement de paquets de tâches sur une grille

Résumé : Dans cet article, nous présentons un algorithme totalement distribué permettant de partager équitablement les ressources d'une grille entre plusieurs applications de type "paquet de tâches". Cet algorithme s'inspire de travaux connexes sur le routage multi-chemin dans les réseaux de communication. Parmi les propriétés intéressantes de cet algorithme, on peut noter sa flexibilité quant au choix du critère d'équité et son optimalité. De plus, cet algorithme n'utilise que des informations locales à chacune des machines participant à l'ordonnancement. Une adaptation naïve des techniques de routage multi-chemin est cependant instable et inefficace. Nous proposons une normalisation simple et efficace permettant de résoudre ces problèmes. Cette normalisation est justifiée par une étude détaillée des différences avec le problème classique de routage multi-chemin. Nous démontrons son efficacité à l'aide de l'analyse détaillée d'une simulation simple.

Mots-clés : Ordonnancement, calcul distribué, optimisation lagrangienne

1 Introduction

In this article, we consider the problem of concurrently scheduling multiple applications on a heterogeneous network of computers such as grid. When multiple applications concurrently run on heterogeneous platforms, inefficiencies generally occur [1]. Hence, a form of cooperation should be induced. On the other hand, centralized allocation schemes generally cannot be used in large systems, as they require a full knowledge of the system at hand. This is why one should seek fully decentralized solutions.

The work most closely related to ours is [2] where centralized and decentralized solutions are proposed and analyzed, and some fairness is insured. However, their analysis is limited to the notion of max-min fairness, which is known to make poor usage of resources [3]. In addition, their decentralized solutions are heuristics that sometimes fail to achieve the desired objective function. In contrast, the algorithm presented in this paper allows to obtain a wide class of allocations, and in particular the α-fair one, with parameter α being a fine tuning between performance and fairness. Finally, the algorithm has convergence guarantees.

We target at methods similar to the ones used in flow control in communication networks, which are based on a coupled source / link algorithm [4]. At each time epoch, the links compute a virtual price based on their availability (generally measured via the queue size or the amount of dropping packets), while the sources respond by dynamically adjusting their rates so as to optimize a given utility function. In grid computing, however, we aim at fully decentralized solutions, by allowing the source algorithm to be executed locally at each node of the system. In this paper, we show that only local information is required at each node and that the source algorithm itself can be fully decentralized.

The contribution of this paper is threefold:

- We adapt an algorithm developed in the context of multi-path routing in communication network [5] to take into account the specificities of grid computing (e.g., the CPU requirements of applications or the fact that "classical" rate adaptation proposed are unstable in this context);
- We propose a decentralized version of the source algorithm;
- We explain the convergence difficulties raised by the differences with the classical multi-path routing problem through a detailed analysis of a simple simulation.

The rest of this paper is organized as follows. We describe in Section 2 the platform and application model we use in this article. We also formally state and justify the optimization problem to be solved. Then, we present related work in Section 3 and specifically the flow control problem in multi-path network. We present the two algorithms proposed by Wang, Palaniswami and Low [5] in this context and their theoretical foundations. The many differences required in the adaptation of one of the algorithms to our framework are highlighted in Section 4. In particular, a scaling mechanism to ensure stability around the optimal solution is proposed. Section 5 is devoted to the presentation of a numerical validation through simulations of a prototype of this algorithm. We present a detailed analysis of the behavior of the algorithm and show that a naive adaptation of the previous technique is unstable and inefficient. Fortunately, the simple scaling mechanism proposed in the previous section proves effective. We finally conclude this article with prospective remarks in Section 6.

2 Framework and Models

We use throughout this paper similar notations to those used in [6] and [5].

2.1 Platform Model

The target architectural framework is represented by a *platform graph*, i.e., a node-weighted edge-weighted graph $G = (N, E, W, B)$, as illustrated in Figure 1. Each node $P_n \in N$ represents a computing resource that can deliver W_n floating-point operations per second. Each edge $e_{i,j} : (P_i \rightarrow P_j)$ is labeled by a value $B_{i,j}$ which represents the bandwidth between P_i and P_j. We assume a linear-cost communication and computation model, hence it takes $X/B_{i,j}$ time units to send a message of size X from P_i to P_j.

We assume that all W_i are positive rational numbers. $W_i = 0$ means that P_i has no computing power but can still forward data to other processors. Similarly, we assume that all $B_{i,j}$ are positive rational numbers (or equal to 0 if there is no link between P_i and P_j).

The operation mode of the processors is the *full overlap, multi-port model* for both incoming and outgoing communications. In this model, a processor node can simultaneously receive data from all his neighbors, perform some (independent) computation, and send data to all its neighbors at arbitrary rate while respecting the resource constraints (i.e., the bandwidth and processing speed bounds).

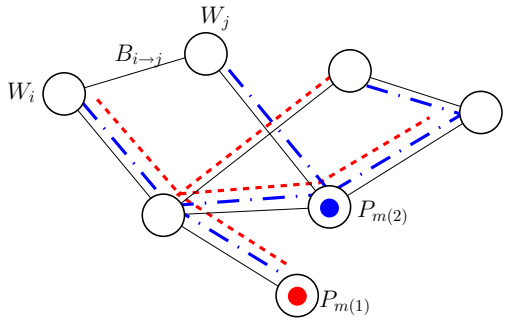

Figure 1: A graph labeled with node (computation) and edge (communication) weights. Two application deployments with different sources.

Altogether, the model is quite simple: linear costs for computing and communicating, full computation-communication overlap, and multi-port model. Note that this framework also comprises the bounded multi-port extension [7] where each node has an additional specific bandwidth bound. This extension simply amounts to slightly change the graph. However, no specific assumption is made on the interconnection graph, which may well include cycles and multiple paths, and contention over communication links is taken into account.

2.2 Application Model

We consider K applications, A_k, $1 \leqslant k \leqslant K$. Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k. Each application is composed of a set of independent, equal-sized tasks. We can think of each A_k as a bag of tasks, where the tasks are files that require some processing. A task of application A_k is called a task of *type* k. We assume one can express the computational requirements of tasks as a number of floating-point operations, and we let w_k be the amount of computation (in floating point operations) required to process a task of type k. Similarly, b_k is the size (in bytes) of (the file associated to) a task of type k. We assume that the only communication required is outwards from the master nodes, i.e., that the amount of data returned by the worker is negligible. We also assume that each application A_k is deployed on the platform as a tree. This assumption is reasonable as this kind of hierarchical deployment is used by many grid services [8]. Therefore if an application k wants to use a node P_n, all its data will use a single path from $P_{m(k)}$ to P_n denoted by $(P_{m(k)} \rightsquigarrow P_n)$. If

no such path exists or if application k cannot access node n (e.g., for administrative reasons), then $(P_{m(k)} \rightsquigarrow P_n)$ is empty.

2.3 Steady-State Scheduling

As each application has an unlimited supply of tasks, each application should aim at maximizing its average number of task processed per time-unit (the *throughput*). When the number of tasks is very large, optimizing the steady-state throughput enables to derive periodic asymptotically optimal schedules for the makespan [9]. In our setting where each application has a very large number of tasks, we should thus try to optimize the steady-state throughput of each application. We refer the reader to [6] for more details on steady-state scheduling. In this article, we denote by $\varrho_{n,k}$ the average number of tasks of type k executed by P_n per time unit. It has been shown in [6] that feasible steady-state rates (i.e., feasible $\varrho_{n,k}$'s) could be used to derive efficient autonomous and dynamic schedules. That is why in this article, we only focus on determining such rates in a fully decentralized way.

2.4 Utility

The throughput of application k at the steady state can be noted $\varrho_k = \sum_{n \in N} \varrho_{n,k}$. Let $U_k(\varrho_k)$ be the utility associated to application k. We aim at maximizing $\sum_{k \in K} U_k$. It has been shown that different values of U_k leads to different kind of fairness [10]. Typically, $U_k(\varrho_k) = \log(\varrho_k)$ (proportional fairness) or $U_k(\varrho_k) = \varrho_k^\alpha/(1-\alpha)$ (α-fairness).

In this article we focus on proportional fairness but the algorithm we present can be straightforwardly adapted to α-fairness.

2.5 Optimization Problem

We aim at finding $(\varrho_{n,k})_{1 \leqslant k \leqslant K, 1 \leqslant n \leqslant N}$ that maximizes $\sum_k \log(\varrho_k)$ under the constraints:

$$\varrho_k = \sum_n \varrho_{n,k} \tag{1}$$

$$\forall n, \sum_k \varrho_{n,k} w_k \leqslant W_n \tag{2}$$

$$\forall (P_i \to P_j), \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \leqslant B_{i,j} \tag{3}$$

Not that the first constraint (1) is not a real constraint of the system but is only introduced for convenience of notations. Constraint (2) simply states that the computation capacity of processor n cannot be exceeded. Similarly, constraint (3) states that the network capacity of link $(P_i \rightarrow P_j)$ cannot be exceeded.

Note that this framework is very general as it does rely neither on the assumptions that all applications originate from the same location nor that all processors are available to all applications (such restrictions can seamlessly be incorporated in the previous equations).

3 Related Work

3.1 Flow Control in Multi-path Network

Typically in networking environments, the optimal flow control can be written as $\max_{\varrho} \sum_s U_s(\varrho_s)$ under the constraints

$$\varrho_k = \sum_n \varrho_{n,k} \tag{4}$$

$$\forall (P_i \rightarrow P_j), \sum_k \sum_{\substack{\text{routes } n \text{ s.t.} \\ (P_i \rightarrow P_j) \in n}} \varrho_{n,k} \leqslant B_{i,j} \tag{5}$$

The first constraint (4) is characteristic of multi-path scenarios, that is to say systems in which each packet sent can follow its own path. The second constraint (5) represents the capacity constraints. Each connection k then has n_k possible routes. Additionally, we suppose that U_s are continuous, increasing and strictly concave utility functions. Wang, Palaniswami and Low [5] specifically address this problem with the additional constraint that each flow has some minimum and maximum requirements: $\forall s, \exists m_s, M_s, m_k \leqslant \varrho_k \leqslant M_k$. This kind of constraints is not really relevant in our context so, for sake of clarity, we only present simplified versions of the equations and algorithms proposed by [5].

Such a problem can be solved using the Lagrange method. A Lagrangian multiplier μ is introduced for each constraint (5). The Lagrangian can thus be written:

$$L(\varrho, \mu) = \sum_{k \in K} U_k \left(\sum_{n=1}^{n_k} \varrho_{n,k} \right) + \sum_{(P_i \rightarrow P_j)} \mu_{i,j} \left(B_{i,j} - \sum_k \sum_{\substack{\text{routes } n \text{ s.t.} \\ (P_i \rightarrow P_j) \in n}} \varrho_{n,k} \right).$$

The constraint set is convex and the U_s are strictly concave. Hence, the original problem is equivalent to solve:

$$\max_{\varrho \geqslant 0} \min_{\mu \geqslant 0} L(\varrho, \mu). \tag{6}$$

Using Slater's conditions, we know that at the optimum, we have $\varrho_{n,k} \cdot \frac{\partial L}{\partial \varrho_{n,k}} = 0$. Hence for any (n,k), we have

$$U'_k(\varrho_k) = \sum_{(P_i \to P_j) \in \text{ route } n} \mu_{i,j} \text{ if and only if } \varrho_{n,k} > 0. \tag{7}$$

$\mu_{i,j}$ is generally called shadow price for resource $(P_i \to P_j)$ and thus $\sum_{(P_i \to P_j) \in \text{route } n} \mu_{i,j}$ can be seen as the price for flow k to use route n. Let us denote it p_k^n. Computing the shadow prices is generally done by applying a gradient method to solve problem (6).

Equation (7) shows that, at optimum, all routes have identical cost p_k and thus

$$\varrho_k^* = \sum_n \varrho_{n,k}^* = U'^{-1}(\min_n p_k^n).$$

The problem is that this approach leads to an expression of the global throughput ϱ_k only, but not the throughputs per path it results from (i.e., the values of $\varrho_{n,k}$). Additionally, the min function is not twice differentiable and a gradient algorithm based on this approach may oscillate and exhibit convergence problems. This is due to the fact that the original optimization problem is *strictly* convex in any of the ϱ_k "variables," but not with respect to the $\varrho_{n,k}$, where it is only convex.

In [5], two alternative algorithms are proposed. We briefly present and comment on these in the remaining of this section.

3.1.1 First Algorithm

The idea of the first algorithm is to modify the objective function so as to make it strictly concave in each of the $\varrho_{n,k}$. Consider the alternative modified objective:

$$\max_{\varrho \geqslant 0, \widetilde{\varrho} \geqslant 0} \sum U_{k=1}^K \left(\sum_{n=1}^{n_k} \varrho_{n,k} \right) - \sum_k \sum_n \frac{1}{2} (\varrho_{n,k} - \widetilde{\varrho}_{n,k})^2,$$

where $\widetilde{\varrho}_{n,k}$ is an augmented variable.

This is a very elegant trick. Indeed, at the optimum, $\widetilde{\varrho}_{n,k} = \varrho_{n,k}$ and hence the solution of this optimization problem (subject to the former constraints) is the

same as the original one. Yet, this optimization problem is strictly concave in all the variables $\widetilde{\varrho}_{n,k}$ and $\varrho_{n,k}$ and hence classical gradient methods do converge. A distributed gradient with a fixed step size γ both for the primal (ϱ and $\widetilde{\varrho}$) and the dual (μ) variables leads to Algorithm 1 for the sources and Algorithm 2 for the links.

forall source k **do**
  Receive the path prices p_k^n.
  forall $n \in 1 \dots n_k$ **do**
    $\widetilde{\varrho}_{n,k}(t+1) \leftarrow (1-\gamma)\widetilde{\varrho}_{n,k}(t) + \gamma\varrho_{n,k}(t)$
    $\varrho_{n,k}(t+1) \leftarrow [(1-\gamma)\varrho_{n,k}(t) + \gamma\widetilde{\varrho}_{n,k}(t) + \gamma(U_k'(\varrho_k(t)) - p_k^n(t))]^+$
    $\varrho_k(t+1) \leftarrow \sum_{n=1}^{n_k} \varrho_{n,k}(t+1)$
  Send $\varrho_{n,k}(t+1)$ to all links in path n.

Algorithm 1: Flow control algorithm for source k

forall Link $(P_i \rightarrow P_j)$ **do**
  Receive $\varrho_{n,k}$ for all route n of connection k, such that $(P_i \rightarrow P_j) \in n$ and aggregate these rates in $\varrho_{i,j}$:
  $\mu_{i,j}(t+1) \leftarrow [\mu_{i,j}(t) + \gamma(\varrho_{i,j}(t) - B_{i,j})]^+$
  Send $\mu_{i,j}(t+1)$ to all sources k using $(P_i \rightarrow P_j)$.

Algorithm 2: Flow control algorithm for link $(P_i \rightarrow P_j)$

3.1.2 Second Algorithm

The second proposed algorithm in [5] is based on a sub-gradient method, on the original optimization problem. The source algorithm is split into a *flow control* problem that computes the total flow ϱ_k for each connection k and a *routing problem* that, given ϱ_k, determines how to split the traffic between the different routes.

The flow control at source k computes

$$\varrho_k^* = U'^{-1}(p_k^*), \text{where } p_k^*(t) = \min_{\text{routes } n} p_k^n(t)$$

The routing algorithm works as follows:

1. Updates all rates:

$$\forall n, \varrho_{n,k}(t+1) = [\varrho_{n,k}(t) - \gamma(p_k^n(t) - p_k^*(t))]^+ . \quad (8)$$

2. Then, each source picks any route j that has minimum price and update its rate to

$$\varrho_{j,k}(t+1) = \left[\varrho_k(t+1) - \sum_{n \neq j} \varrho_{n,k}(t+1) \right]^+ \quad (9)$$

Thus, for each connection, the use of all paths that cost more than the minimum price is gradually decreased (from Equation (8)). This insures that at the equilibrium, only routes of minimum costs are used.

Additionally, at each time epoch, when a connection has several routes of equal minimum cost, the rate of one route is adjusted so that $\sum_n \varrho_{n,k}$ remains equal to ϱ_k.

The writing of algorithm being very similar to Algorithm 1, the details are not given here. The interested reader would refer to [5] for a full description.

3.2 Discussion

As can be straightforwardly noted, the optimal bag-of-task application scheduling on grid, as described in Section 2.5 is very similar to the optimal flow control problem in multi-path routing. For this reason, we study in this paper how to adapt it to our context. The main differences between the two frameworks are developed in the next paragraphs.

3.2.1 Routes vs Slaves

Grids are generally high-level infrastructures where security and access restrictions are important. Thus there is generally a single way to go from a location to another. This induces hierarchical deployment that are used by many grid services and a unique (canonical) route from a master to a specific slave, in contrary to multi-path routing.

3.2.2 Lagrange Multipliers

Another difference lies in the additional constraint (2) on computations. This new constraint does not fundamentally change the problem. It only amounts to add an other Lagrangian multiplier λ similar to μ. Also, in our context there is no need

to consider here minimal and maximal requirements (m_k and M_k respectively) as introduced in [5]. These differences only imply simple modifications of the algorithms.

3.2.3 Topology

The topology of the systems is very different, which has a non-negligible impact on the convergence rate. In grids, the master-slave pairs are analogous to routes in the flow control problem, and applications are analogous to connections. Compared to the flow control problem, there is thus a huge number of "routes" and a small number of "sources." This may have some important impact on the convergence rate.

3.2.4 Decision Nodes

A subsequent difference lies in the decision points. While in networking context, sources adapt and choose their transmission rate, in grids, we would like the intermediate nodes (between a master and each of its slave) to adjust the rates, so as to prevent overloading the master. Hence the newly proposed source algorithm should itself be decentralized.

4 Proposed Algorithm and Important Properties

The Lagrangian is written as follow:

$$L(\varrho,\widetilde{\varrho},\lambda,\mu) = \sum_{k\in K} U_k\left(\sum_i \varrho_{i,k}\right) + \sum_i \lambda_i\left(W_i - \sum_k \varrho_{i,k}w_k\right) - \sum_{k\in K}\sum_i \frac{1}{2}(\varrho_{i,k}-\widetilde{\varrho}_{i,k})^2 + \sum_{(P_i\to P_j)} \mu_{i,j}\left(B_{i,j} - \sum_k \sum_{\substack{n \text{ such that} \\ (P_i\to P_j)\in(P_{m(k)}\rightsquigarrow P_n)}} \varrho_{n,k}b_k\right).$$

For sake of simplicity, we first introduce the following variables η and σ:

$$\begin{cases} \eta_k^n &= \displaystyle\sum_{(P_i\to P_j)\in(P_{m(k)}\rightsquigarrow P_n)} \mu_{i,j}, \\ \sigma_k^n &= \displaystyle\sum_{\substack{n \text{ such that} \\ (P_i\to P_j)\in(P_{m(k)}\rightsquigarrow P_n)}} \varrho_{n,k}. \end{cases}$$

That is, η_k^n is the aggregate price per Mb to send data from $P_{m(k)}$ to P_n and σ_k^n is the total amount of tasks by application k sent from $P_{m(k)}$ to P_n and all its descendants. These variables can be computed with a simple propagation algorithm:

$$\begin{cases} \eta_k^j & = \eta_k^i + \mu_{i,j} \\ \sigma_k^i & = \varrho_{i,k} b_k + \displaystyle\sum_{j \text{ s.t.} (P_i \to P_j)} \sigma_k^j \end{cases}$$

4.1 Algorithm

Using a fixed-size γ step gradient (where the γ value may depend on the considered variable), we derive the simple following updates for λ, μ, ϱ and $\widetilde{\varrho}$:

$$p_k^i(t+1) \leftarrow b_k \eta_k^i(t) + w_k \lambda_i(t) \tag{10}$$

$$\widetilde{\varrho}_{i,k}(t+1) \leftarrow (1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{i,k}(t) + \gamma_{\widetilde{\varrho}} \varrho_{i,k}(t) \tag{11}$$

$$\varrho_{i,k}(t+1) \leftarrow [(1-\gamma_\varrho)\varrho_{i,k}(t) + \gamma_\varrho \widetilde{\varrho}_{i,k}(t) + \gamma_\varrho (U_k'(\varrho_k(t)) - p_k^i(t))]^+ \tag{12}$$

$$\varrho_k(t+1) \leftarrow \sigma_k^{m(k)}(t+1) \tag{13}$$

$$\lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda \left(\sum_k w_k \varrho_{i,k} - W_i\right)\right]^+ \tag{14}$$

$$\mu_{i,j}(t+1) \leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \left(\sum_k b_k \sigma_k^i - B_{i,j}\right)\right]^+ \tag{15}$$

We assume that on each node i there is a process for application k who is responsible for $\varrho_{k,i}$, $\varrho_{k,i}$, $\eta_{k,i}$ and $\sigma_{k,i}$. Each link is responsible for its own $\mu_{i,j}$ and each node is responsible for its own λ_i. It is easy to see from these equations that all updates can be actually done from local variables. Therefore, the set of equations results in Algorithm 3 and 4.

4.2 Interesting Properties

The new algorithm has many interesting following properties. First, it is fully distributed and its convergence is proved provided a correct choice of γ_λ,γ_μ,γ_ϱ, and $\gamma_{\widetilde{\varrho}}$ (Indeed, it corresponds to the Lagrange optimization of a strictly convex optimization function.) Second, it only requires very simple computations and small communications and thus does waste resources. Last, but not least, this algorithm seamlessly adapts to W_i of $B_{i,j}$ variations and to the arrival of new applications.

forall Agent k on node i **do**
- Get local price λ_i
- Get price to receive from the father μ_i
- Receive η_k^i.
- Send $\eta_k^i + \mu_i$ to all sons.
- Receive σ_k^j from all sons.
- $\sigma_k^i \leftarrow \varrho_{k,i} + \sum_j \sigma_k^j$
- Send σ_k^i to the father.
- Receive ϱ_k and forward it to all sons.
- Give $b_k\sigma_k^i$ and $w_k\varrho_{i,k}$ to the local price manager.
- $p_k^i \leftarrow b_k\eta_k^i + w_k\lambda_i$
- $\varrho_{i,k} \leftarrow [(1-\gamma_\varrho)\varrho_{i,k} + \gamma_\varrho\widetilde{\varrho}_{i,k}(t) + \gamma_\varrho(U_k'(\varrho_k(t)) - p_k^i(t))]^+$
- $\widetilde{\varrho}_{i,k} \leftarrow (1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{i,k} + \gamma_{\widetilde{\varrho}}\varrho_{i,k}$

Algorithm 3: Scheduler algorithm for application k on node i

forall Price update on node i **do**
- Get $b_k\sigma_k^i$ and $w_k\varrho_{i,k}$ from the local scheduler of each application.
- $\mu_{i,j} \leftarrow [\mu_{i,j} + \gamma_\mu\left(\sum_k b_k\sigma_k^i - B_{i,j}\right)]^+$
- $\lambda_i \leftarrow [\lambda_i + \gamma_\lambda\left(\sum_k w_k\varrho_{i,k} - W_i\right)]^+$

Algorithm 4: Pricing algorithm k on node i

4.3 Convergence Issues

In our test, we use $U_k(\varrho_k) = \log(\varrho_k)$ and thus the update of $\varrho_{i,k}$ in Equation 12 writes as follows:

$$\varrho_{i,k}(t+1) \leftarrow \left[(1-\gamma_\varrho)\varrho_{i,k}(t) + \gamma_\varrho\widetilde{\varrho}_{i,k}(t) + \gamma_\varrho\left(\frac{1}{\varrho_k(t)} - p_k^i(t)\right)\right]^+.$$

A small value of ϱ leads to huge updates and thus to severe oscillations. As mentioned in [5], it is possible to use independent step sizes for the $\widetilde{\varrho}$ part and for the price p_k^i part and to normalize this update as follows:

$$\varrho_{i,k}(t+1) \leftarrow \left[(1-\gamma_\varrho^{(1)})\varrho_{i,k}(t) + \gamma_\varrho^{(1)}\widetilde{\varrho}_{i,k}(t) + \gamma_\varrho^{(2)}\left(1 - \varrho_k(t).p_k^i(t)\right)\right]^+. \quad (16)$$

Our experiments show that this normalization allows to avoid division by 0 that often occurred with the previous update scheme but that it is unfortunately not sufficient to damp oscillations. This is due to the fact that updating ϱ has an impact on the prices λ and μ, which in turn impact on the ϱ's update. The second update of ϱ should have the same order of magnitude (or be smaller) as the first one to avoid numerical instabilities that prevent convergence of the algorithm.

Let us assume that we have reached the equilibrium. Further assume that the price λ_i of P_i is increased by $\Delta\lambda_i$. From Equation (16), we derive that such an increase incurs a variation $\Delta\varrho_{i,k}$ of $\varrho_{i,k}$:

$$\Delta\varrho_{i,k} = -\gamma_\varrho^{(2)} w_k \Delta\lambda_i \varrho_k.$$

In turn, from Equation (14), we see that such a variation incurs a variation of λ_i:

$$\sum_k \gamma_\lambda . w_k . \Delta\varrho_{i,k} = \Delta\lambda_i . \left(\sum_k \gamma_\lambda . \gamma_\varrho^{(2)} w_k^2 \varrho_k \right).$$

Thus, the solution of our gradient is stable only if

$$\sum_k \gamma_\lambda . \gamma_\varrho^{(2)} w_k^2 \varrho_k < 1.$$

Therefore, Equation (14) should be replaced by

$$\lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{i,k} - W_i}{\sum_k w_k^2 \varrho_k} \right]^+ \tag{17}$$

and Equation (15) should be replaced by

$$\mu_{i,j}(t+1) \leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^i - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} b_k^2 \varrho_k} \right]^+ . \tag{18}$$

Note that this scaling does not even require additional aggregation as all processors already receive ϱ_k to perform the update of ϱ (Equation (16)). Such a scaling is not really needed in the classical multi-path routing problem.

5 Numerical Results

In this section, we present an experimental evaluation of the previous algorithms (the unscaled version versus the scaled one). Section 5.1 details how our simulations were conducted. The analysis of the unscaled algorithm (using Equations (14) and (15)) is presented in Section 5.2. The analysis of the scaled version (using Equations (17) and (18)) is presented in Section 5.3).

5.1 Evaluation Methodology

5.1.1 Implementation

We implemented our algorithm using the SimGrid simulator [11]. It has been shown in [6] that feasible steady-state rates (i.e., feasible $\varrho_{n,k}$'s) could be used to derive efficient autonomous and dynamic schedules. That is why we only focus on determining such rates in a fully decentralized way. In the current implementation, all agents use synchronous communications and thus, each step requires a complete traversal of the deployment tree.

Unless otherwise stated, we use the same default step sizes and initial values in all our experiments. These values are summarized in Table 1.

Table 1: Step parameters

variable	$\gamma_{\widetilde{\varrho}}$	$\gamma_{\varrho}^{(1)}$	$\gamma_{\varrho}^{(2)}$	γ_{λ}	γ_{μ}
value	0.02	0.02	100.0	5.10^{-3}	5.10^{-4}
variable	ϱ_{init}	λ_{init}	μ_{init}		
value	6.10^{5}	2.10^{-2}	2.10^{-2}		

5.1.2 Platform Generation

We performed a first set of tests on a simple platform consisting of only 5 nodes (see Figure 2. For ease of interpretation, we assumed a homogeneous set of nodes and links, although neither the algorithm nor our prototype requires that.

5.1.3 Application Generation

An application is mainly characterized by its *communication-to-computation ratio* (CCR). We used three kinds of applications of respective (b, w): $(1000, 5000)$, $(2000, 800)$, and $(1500, 1500)$. We considered that each application originates from

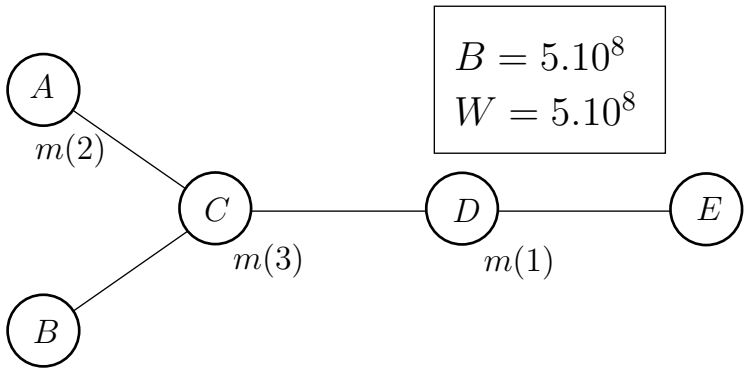

Figure 2: Simple platform topology. All applications originate from different nodes.

a different master, as represented on Figure 2. Namely, application 1, 2 and 3 are hosted by nodes D, A and C respectively.

5.1.4 Validation

Determining whether such iterative algorithms have converged is generally not easy. In our experiments, we assess the convergence of our algorithms using SDP (Semi-Definite Programming). SDP is a mathematical program which solves the minimization problem of a linear combination of variables subject to the constraint of positive semi-definiteness of some general symmetric matrix whose entries are either variables or constants.

It has been shown that the fair allocation of resources subject to linear constraints can be expressed in a SDP program [12]. As this program can be solved in polynomial time, it can provide a quick and reliable, yet centralized, validation of our numerical results.

For each of the simulations presented below, we compared the obtained solutions to the ones provided by the free open source SDP solver called DSDP[1].

5.2 Unscaled Version

Our simulations show that, even in this simple case study like the one presented in this section, step sizes (for ϱ, λ and μ) cannot be properly set.

[1] Available at http://www-unix.mcs.anl.gov/DSDP/

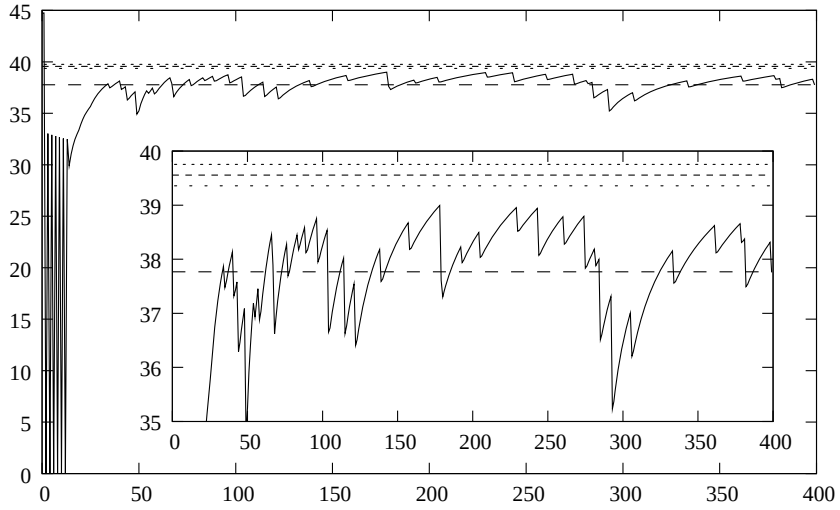

Figure 3: Obtained values of the objective function $\sum \log \varrho$ for the unscaled version and comparison with the optimal one. Different orders of magnitudes in the Lagrangian multipliers lead to numerical instabilities and global inefficiencies.

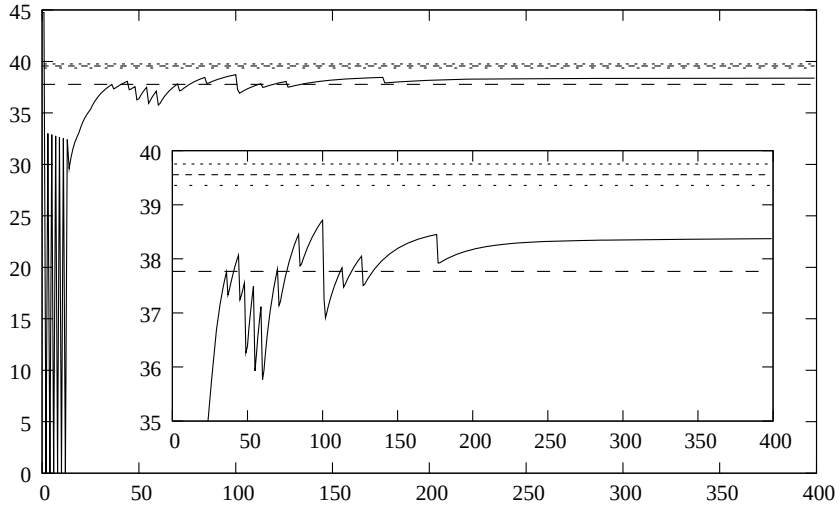

Figure 4: Obtained values of the objective function $\sum \log \varrho$ for the unscaled version and comparison with the optimal one. Smaller step sizes ($\gamma_\lambda = 5.10^{-4}$, $\gamma_\lambda = 5.10^{-5}$, $\gamma_\varrho^{(2)} = 1.0$) help reducing the oscillations and erratic values but incur a very slow convergence.

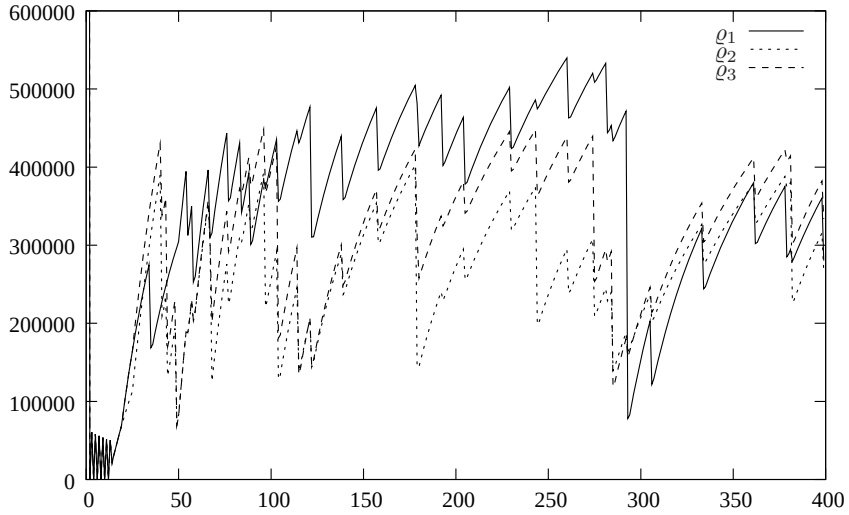

Figure 5: Throughput of each of the three applications during 400 iterations: between two iterations, a decrease or increase of magnitude five or more can happen.

In our experiments, we considered a set of 400 iterations. We observed that the step sizes for $\gamma_{\widetilde{\varrho}}$ and $\gamma_{\varrho}^{(1)}$ did not impact the convergence of the algorithm. On the contrary, the step size $\gamma_{\varrho}^{(2)}$ of ϱ was of upmost importance.

We represent the influence of this parameter on Figures 3 and 4. On each ones are represented 4 horizontal lines. The upper one is the optimal value, as computed by the SDP program while the three others represent the 99.5%, 99% and 95% of the optimal value. For relatively large value of $\gamma_{\varrho}^{(2)}$ (e.g., $\gamma_{\varrho}^{(2)} = 100$), the algorithm proved to be unstable, each update on the values being too large. Large updates can be avoided by using a small step size (for instance $\gamma_{\varrho}^{(2)} = 0.1$, see Figure 4). Yet, although more stable, this scheme does not converge well.

This being said, let us look at the updates on the throughput of each application (in the situation where $\gamma_{\varrho}^{(2)} = 100$). We note that not only do the objective functions fluctuate, but also the throughputs of each application, as represented in Figure 5.

To understand this behavior, let us recall that the throughput of each application is actually the sum of the rates on each of the 5 paths it follows: $\varrho_k = \sum \varrho_{i,k}$. Figure 6 shows how ϱ_1 decomposes. We can note that the different $\varrho_{i,1}$ have exactly the same shape, and only varies by the instants they are reset to zero. These time epochs actually correspond to sudden jumps in the prices.

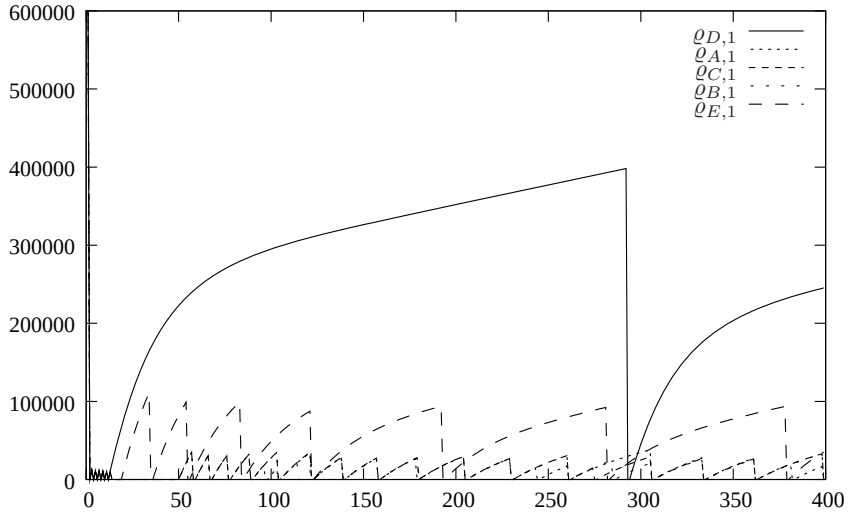

Figure 6: The throughput of application seen as the sum of the rates through the different paths.

Indeed, a closer look at the system indicates erratic values of the prices in each node and link. Indeed, for each resource, the prices are null except for a finite number of points, where the values can be arbitrarily large (Figure 7).

Hence, each application steadily increases it throughput while the prices of the resources it uses are null. Then, at some point a resource becomes saturated and its price suddenly "jumps," leading to a null throughput of all applications that use it. Then the process reiterates, which explains that the system cannot converge. In equation (14) it is easy to see that the price update is directly proportional the saturation of the resource. This saturation has the same order of magnitude as the $w_k\varrho_k$ whereas the price has the same order of magnitude as the inverse of the to the ϱ_k's, which explains why such an unstable in inherent to this algorithm (regardless of the values of γ_λ and γ_μ).

5.3 Scaled Version

We ran the simulations on the same platform with the scaled version of the algorithm. The results on the objective function are presented in Figure 8. We clearly see from these curves how quickly the oscillations vanish with each iteration and how the algorithm converges to the optimal solution.

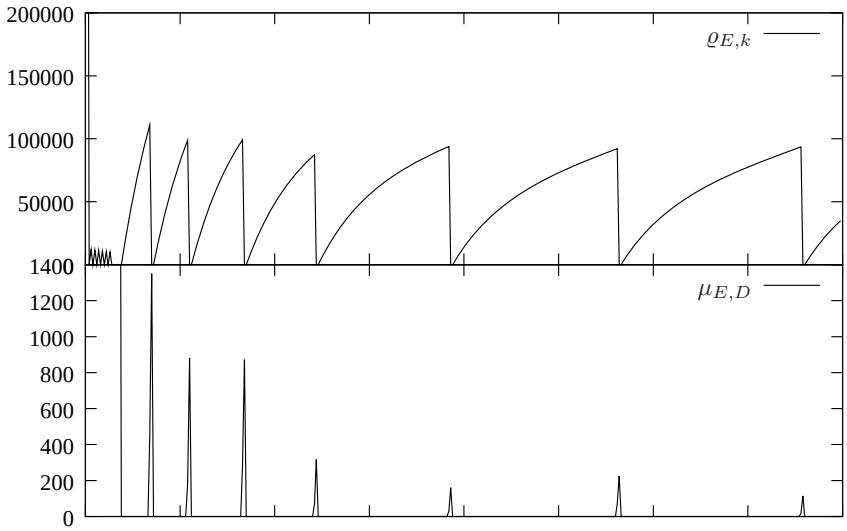

Figure 7: Correlation between the throughput of an application on a given route and the price it experiences.

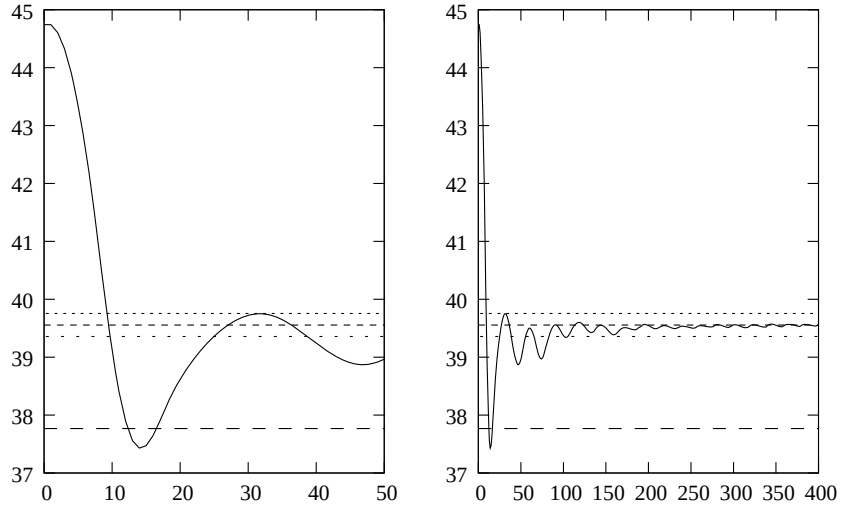

Figure 8: Scaled version of the algorithm: the oscillations, due to a really badly chosen initialisation value quickly vanish (left graph). The algorithm almost instantly reaches a decent value (5% of the optimal value after 17 iterations), and relatively quickly to a good value (1% of the optimal guaranteed after 83 iterations) (right plot).

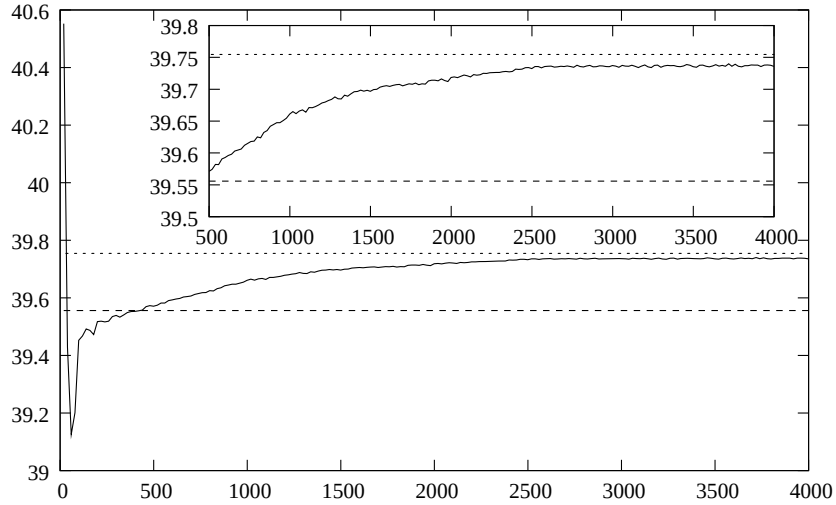

Figure 9: Scaled version of the algorith - high number of iterations: after 498 iterations, the performance remains higher than 99.5% of the optimal and still further increase with the number of iterations (Nota: each point of the curve is obtained as the average of 20 iterations).

We can observe from Figure 9 that the iteration number can be actually chosen so as to obtain arbitrarily good performance, as opposed to what was observed with the unscaled version.

Let us look at the throughput achieved by each application (Figure 10). As with the objective function, we observe than after a few tens of iterations, the values obtained by our algorithm are fairly closed to the optimal one. Yet, the convergence is not as quick. After 2000 iterations, an application can still be at approximately 5% of it equilibrium value. However, the system very quickly converges to rate that make good usage of resources. More interestingly, after the "initialization phase" (the first few tens of iterations), we do not observe oscillations or large updates, which is a very desired feature when it comes to implementation.

Let us finally compare the prices evolution (Figure 11). As opposed to the unscaled version, where they were either null or with high values, the prices now have small, yet strictly positive values and continuously change with the iteration number. This reflects an harmonious utilization of resources. Furthermore, we note that, although their oscillations are synchronized during the first steps of the algorithm (due to the synchronous arrival of the applications in the system), their curve quickly disynchronized, as opposed to the unscaled version.

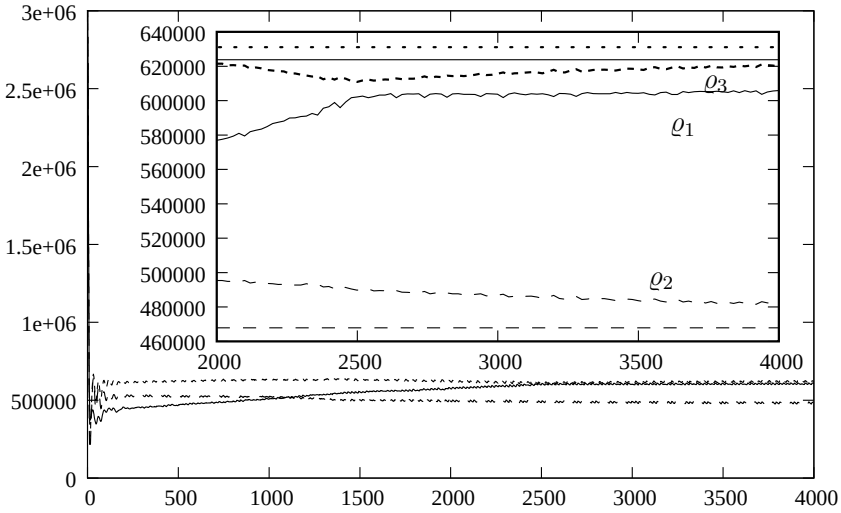

Figure 10: Convergence of ϱ_i with $i = 1..3$: no more oscillations occur.

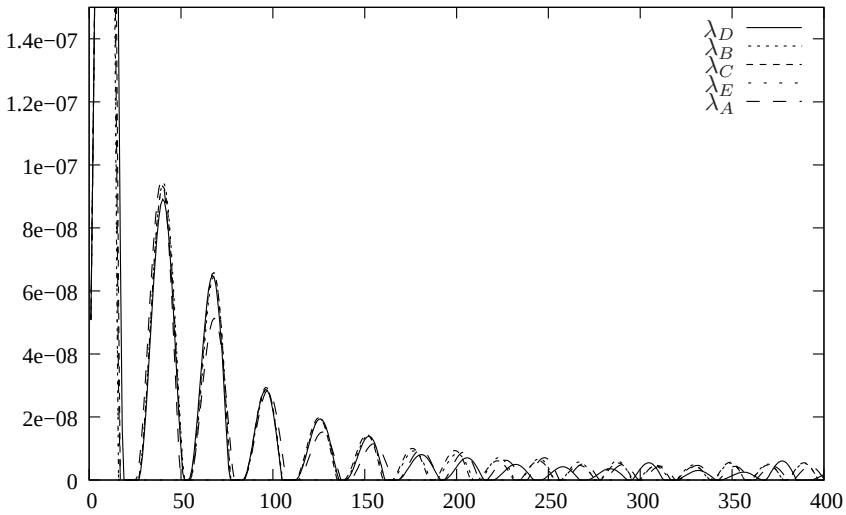

Figure 11: Ressource prices in scaled algorithm: prices evolve smoothly. As the number of iterations increase, they converge to zero while remaining positive, meaning that the ressources they refer to is neither under utilized nor overloaded.

6 Conclusion

In this article we have explained how to adapt an algorithm designed in the context of multi-path routing in communication network [5] to a multiple Bag-of-tasks application scheduling problem. As demonstrated by our simulations and our analyses, a "naive" adaptation is ineffective and a simple novel scaling scheme has been proposed to damp oscillations in this new context. The need for this scaling is mainly due to the fact that the resource usage is not homogeneous (each application has its own w_k and b_k). We think that the scaling scheme proposed is simple enough to be of practical interest and will work on more complex examples than the one presented in this article (our preliminary simulation results tend to confirm this hypothesis but a complete study is underway). We can however think of a situation where this scaling could be insufficient to ensure a fast convergence. When the optimal throughput of the applications do not have the same order of magnitude, it may be necessary for each application to have its own step size $\gamma_\varrho^{(2)}$. Ultimately, a scaling on $\gamma_\varrho^{(2)}$ so that each update has the same order of magnitude as the corresponding $\varrho_{i,k}$ would ensure a fast convergence in all situations but we are still investigating this issue.

References

[1] A. Legrand and C. Touati, "Non-cooperative scheduling of multiple bag-of-task appplications," in *Proceedings of the 25th Conference on Computer Communications (INFOCOM'07)*, Alaska, USA, May 2007.

[2] O. Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, and Yves Robert, "Centralized versus distributed schedulers multiple bag-of-task applications," in *International Parallel and Distributed Processing Symposium IPDPS'2006*. IEEE Computer Society Press, 2006.

[3] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[4] S. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 525–536, 2003.

[5] W.-H. Wang, M. Palaniswami, and S. Low, "Optimal flow control and routing in multi-path networks," *Performance Evaluation*, vol. 52, pp. 119–132, 2003.

[6] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus distributed schedulers multiple bag-of-tasks applications," *IEEE Trans. Parallel Distributed Systems*, vol. 19, no. 5, pp. 698–709, May 2008.

[7] B. Hong and V. K. Prasanna, "Adaptive allocation of independent tasks to maximize throughput," *IEEE Trans. Parallel Distributed Systems*, vol. 18, no. 10, pp. 1420–1435, Oct. 2007.

[8] E. Caron and F. Desprez, "Diet: A scalable toolbox to build network enabled servers on the grid," *International Journal of High Performance Computing Applications*, vol. 20, no. 3, pp. 335–352, 2006.

[9] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Steady-state scheduling on heterogeneous clusters: Why and how?" in *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2004*. IEEE Computer Society Press, 2004, p. 171a (8 pages).

[10] C. Touati, E. Altman, and J. Galtier, "Generalized Nash bargaining solution for bandwidth allocation," *Computer Networks*, vol. 50, no. 17, pp. 3242–3263, Dec. 2006.

[11] A. Legrand, M. Quinson, K. Fujiwara, and H. Casanova, "The SimGrid project - simulation and deployment of distributed applications," in *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*. IEEE Computer Society Press, 2006, pp. 385–386.

[12] C. Touati, E. Altman, and J. Galtier, "Generalized Nash bargaining solution for bandwidth allocation," *Computer Networks*, vol. 50, no. 17, pp. 3242–3263, Dec. 2006.

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

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

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