From Flow Control in Multi-path Networks to Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin, Sascha Hunold, Arnaud Legrand, Corinne Touati

▶ **To cite this version:**

Rémi Bertin, Sascha Hunold, Arnaud Legrand, Corinne Touati. From Flow Control in Multi-path Networks to Multiple Bag-of-tasks Application Scheduling on Grids. [Research Report] RR-7745, INRIA. 2011, pp.26. inria-00627532

HAL Id: inria-00627532

https://inria.hal.science/inria-00627532

Submitted on 28 Sep 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

From Flow Control in Multi-path Networks to Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin — Sascha Hunold — Arnaud Legrand — Corinne Touati

N° 7745

Septembre 2011

Domaine 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7745--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

From Flow Control in Multi-path Networks to Multiple Bag-of-tasks Application Scheduling on Grids

Rémi Bertin , Sascha Hunold , Arnaud Legrand , Corinne Touati

Domaine : Réseaux, systèmes et services, calcul distribué
Équipes-Projets MESCAL

Rapport de recherche n° 7745 — Septembre 2011 — 26 pages

Abstract: A very large number of applications that are currently deployed on large scale distributed systems such as grids or volunteer computing systems are Bag-of-Tasks (BoT) applications. Up until now, simple mechanisms have been used to ensure a fair sharing of resources amongst these applications. Although these mechanisms have proved efficient for CPU-bound applications, they are known to be ineffective in the presence of network-bound applications.

In this article, we propose a fully distributed algorithm for fairly scheduling BoT applications on a computing grid while respecting resource constraints. This algorithm is inspired by techniques used in flow control mechanisms in multi-path networks. Yet, we prove that the context of BoT scheduling is significantly more difficult in practice and that a set of non-trivial adaptations are required to ensure convergence. We prove their effectiveness through an extensive set of simulations that enables to deeply understand the potential benefits as well as the limitations of this technique in the context of grid computing.

Key-words: Lagrangian optimization, steady-state scheduling, distributed scheduling, grid computing

Du contrôle de flux dans les réseaux multi-chemins à l'ordonnancement en régime permanent d'applications de type *Bag-of-tasks* dans des grilles de calcul

Résumé : Une grande partie des applications actuellement déployées à grande échelle est de type Bag-of-Tasks (BoT), c'est-à-dire qu'elles sont constituées d'un grand nombre de tâches identiques et indépendantes. Jusqu'ici, seulement des mécanismes simples ont été mis en œuvre pour s'assurer d'un partage équitable des ressources entre les applications. Si ces mécanismes ont prouvé leur efficacité dans le cas où les applications sont gourmandes en calcul, leur inefficacité en présence d'applications gourmandes en communications est également connue.

Dans cet article, nous proposons un algorithme complètement distribué pour l'ordonnancement équitable d'applications de type BoT tout en exploitant efficacement l'ensemble des ressources (de communication et de calcul). Cet algorithme s'inspire de techniques qui ont été utilisées dans le domaine du contrôle de flux dans les réseaux multi-chemins. En dépit de leur ressemblance sur le plan théorique, nous montrons que le contexte de l'ordonnancement d'applications BoT dans une grille est significativement plus délicat en pratique que celui du contrôle de flux dans des réseaux multi-chemins. Un ensemble d'adaptations non triviales est nécessaire pour obtenir la convergence et nous montrons leur efficacité à travers un ensemble conséquent de simulations

Nous pensons que l'analyse minutieuse de cet algorithme présentée dans ce document permet de comprendre finement les avantages et les limitations de cette technique dans le contexte des grilles de calcul.

Mots-clés : Optimisation Lagrangienne, ordonnancement en régime permanent, ordonnancement distribué, calcul à grande échelle

1 Introduction

A very large number of applications that are currently deployed on large scale distributed systems such as grids or volunteer computing systems are Bag-of-Tasks (BoT) applications. Up until now, mainly simple mechanisms have been used to ensure a fair sharing of resources amongst these applications. Although these mechanisms have proved efficient for CPU-bound applications, they are known to be ineffective in the presence of network-bound applications. In this article, we propose a fully distributed algorithm for fairly scheduling BoT applications on a computing grid while respecting resource constraints.

We first review in Section 2 network protocol engineering techniques based on Lagrangian optimization and distributed gradient. These techniques have been widely used in the networking community and in particular to propose flow control mechanisms in multi-path networks.

Then, we explain in Section 3 how the flow control problem is similar to the one of fair resource sharing between multiple BoT applications in a grid environment. Although the flow-control algorithm for multi-path networks had only been evaluated in a very limited setting, such technique is very appealing as it allows the choice of a wide variety of fairness criteria and achieves both optimal path selection and flow control.

We show in Section 4 how this technique enables to design a hierarchical and distributed algorithm. This algorithm only requires local information at each worker computing tasks and at each buffer of the network links.

Yet, we demonstrate in Section 5 through a carefully designed set of simulations that a direct adaptation of this protocol to the grid context is effective if and only if all applications are identical:application heterogeneity raises very complex practical convergence issues.

To address heterogeneity, we detail in Section 6 a set of non-trivial adaptations that are required to ensure convergence. In Section 7, we prove their effectiveness in a fully heterogeneous setting through an extensive set of simulations.

We believe that the thorough analysis we conduct in this article enables to deeply understand the potential benefits as well as the limitations of this technique in the context of grid computing.

The contributions of this article can be summarized as follows:

- The proposal of a *fair* and *optimal hierarchical scheduling algorithm* for BoT applications with *arbitrary communication-to-computation ratio*. Popular existing infrastructures do not offer support for applications with such characteristics. The kind of algorithm we propose is thus a first step toward this direction.
- A comprehensive survey on how Lagrangian optimization and gradient descent has been used to design network protocols.
- An experimental proof that although a naive adaptation of the previous technique is effective when all applications are identical, it is bound to fail when applications have different characteristics. BoT scheduling on grid computing platforms is thus significantly more complex than flow control in multi-path networks.

- A set of non-trivial adaptations required to come up with an algorithm whose effectiveness is assessed in a wide variety of scenarios.
- We show how to carry out step-size tuning, which is generally tedious for such algorithms and we experimentally prove that such parameters can be adjusted so that our algorithm is robust to platform topology modification.

2 Related Work

2.1 Resource Sharing Between Multiple Concurrent BoT Applications

In this article, we focus on scheduling multiple BoT applications in steady-state. Steady-state has been introduced to model situations when each application has a very large or an unlimited supply of tasks. In such situations, applications should aim at maximizing their average number of tasks processed per time-unit (the *throughput*). Such objective is both more meaningful and easier to optimize than classical makespan optimization. Furthermore, optimizing the steady-state throughput enables to derive periodic asymptotically optimal schedules for the makespan [1]. We refer the reader to [2] for more details on steady-state scheduling.

This scenario is somehow similar to that addressed by existing systems. For instance BOINC [3] is a centralized scheduler that distributes tasks for participating applications, such as SETI@home, ClimatePrediction.NET, and Einstein@Home. The applications can be very different in nature, e.g., files to be processed, images to be analyzed or matrices to be manipulated. Yet, most of existing systems are client-server oriented and tackle applications with a very small communication-to-computation ratio (CCR). This is a key simplifying hypothesis as it enables to serve clients regardless of their connectivity and avoids server overload. It also enables to rely on very simple sharing mechanisms. For example the BOINC sharing policy is to fairly share on each client the CPU resource among projects to which the volunteer subscribed [4]. It has been proved [5] that such a simplistic and local approach leads to resource waste whenever communication links become critical resources. The OurGrid infrastructure relies on a *tit-for-tat* mechanism inspired by the BitTorrent bandwidth sharing mechanism [6].

In this article however, we propose and study the design of a *fair* and *optimal hierarchical scheduling algorithm* for applications with *arbitrary communication-to-computation ratio.* The Large Hadron Collider Computing Grid (LCG) [7] is a system with such needs. The Large Hadron Collider (LHC) produces roughly 15 Petabytes of data annually that are accessed and analyzed by thousands of scientists around the world. The resulting computation tasks have a much larger communication-to-computation ratio than typical distributed computing computing applications and their efficient management is still an open problem.

The BoT scheduling problem has also been widely studied under less restrictive hypothesis. For example, the scheduling problem is completely different if there are fewer tasks than machines. The most well-known systems for more general situations are APST [8], Nimrod/G [9], Condor [10] and more recently MyGrid [6]. The objective is often to minimize completion time while the main issue is to select resources. Such problems are generally solved with list scheduling heuristics like min-min, sufferage, or similar variations [11, 12]. It is also generally impossible to assume that nodes are reliable and deliver constant computing power. Such uncertainties are at the heart of selection mechanisms and replication is thus often used to avoid waiting for the last tasks of a BoT [13].

2.2 Fairness and Network Protocol Design

In the last decade, the network community has used Lagrangian optimization and distributed gradient techniques to both analyze and design network protocols like TCP (see for example [14, 15]). We start by reviewing the basis of these techniques to understand their key components. Then, we transfer them to the grid context.

Assume we are given a network made of a set of links $\mathcal{L}$ whose capacity is to be shared amongst a set of flows $\mathcal{F}$. Let us denote by ϱ_f the bandwidth allotted to flow f. Fairly and efficiently sharing resources amongst applications has been widely handled in economics through the notion of utility, which is a measure of relative satisfaction of users (or flows here). If we denote by $U_f(\varrho_f)$ the utility associated to flow f when it is allotted a throughput ϱ_f, it is common to aim at maximizing $\sum_{f\in\mathcal{F}} U_f(\varrho_f)$. It has been shown that different choices of U_f leads to different kind of fairness [16]. Common choices are $U_f(\varrho_f) = \log(\varrho_f)$ (proportional fairness [15]) or $U_f(\varrho_f) = \varrho_f^\alpha/(1-\alpha)$ (α-fairness [16], which covers in particular the cases of proportional fairness for $\alpha \to 1$, max-min fairness [17] when $\alpha \to \infty$, social welfare for $\alpha = 0$).

Let us thus assume the network operator has decided to share bandwidth according to some fairness criteria expressed through utility functions U_f. The bandwidth sharing can be written as follows:

$$\begin{array}{l}\max \sum_{f\in\mathcal{F}} U_f(\varrho_f) \\ \begin{cases} \forall l \in \mathcal{L}, & \displaystyle\sum_{f \text{ going through } l} \varrho_f \leqslant B_l \end{cases}\end{array} \tag{1}$$

Solving such an optimization problem in a centralized way would be impracticable. Developing a distributed algorithm has thus received a lot of attention. The main issue is that checking that all constraints are satisfied requires a global coordination, which is very hard to implement. Hopefully, Lagrangian optimization enables us to put the previous problem in a form more amenable to distribution. This is achieved by introducing a dual variable for each constraint and hence for each resource, which we will denote by λ_l. The Lagrangian function is then written as:

$$L(\varrho, \lambda) = \underbrace{\sum_{f\in\mathcal{F}} U_f(\varrho_f)}_{\text{objective function}} + \sum_{l\in\mathcal{L}} \lambda_l \cdot \underbrace{\left(B_l - \sum_{f \text{ through } l} \varrho_f\right)}_{\text{constraints}} \tag{2}$$

The original problem (1) can be safely be rewritten:

$$\max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\varrho, \lambda).$$

Indeed, if an allocation ϱ is unfeasible, then one of the constraints is violated and the inner minimization problem is thus minimized by setting the corresponding

λ_l to $+\infty$. Conversely, if ϱ is a feasible allocation, then the inner minimization problem is minimized by setting the corresponding λ_l to either 0 when the constraints are not tight or to any positive value when the constraint is tight. The objective value is then equal to the original objective function and this formulation of the primal is thus strictly equivalent to our original problem. Under very mild assumptions it can be proven that there is no duality gap, i.e., that

$$\max_{\varrho \geqslant 0} \min_{\lambda \geqslant 0} L(\varrho, \lambda) = \min_{\lambda \geqslant 0} \underbrace{\max_{\varrho \geqslant 0} L(\varrho, \lambda)}_{d(\lambda)}$$

In most cases U_f is chosen to be continuous, increasing and strictly concave and the dual function d is thus a convex function. Solving the original problem is equivalent to find the saddle point of L. The main advantage of such reformulation is that now constraints are very simple ($\varrho \geqslant 0$ and $\lambda \geqslant 0$) and do not require any global coordination. Since both concave maximization problem and convex minimization problems can be solved through gradient descent (see for example [18], chapter 3, on the convergence analysis of descent algorithms), the saddle point is generally obtained making a gradient descent simultaneously for both inner and outer optimization problems. A simple constant step-size (γ_ϱ) ascent on the primal variables simultaneous to a constant step-size (γ_λ) descent on the dual variables leads the following update equations

$$\begin{cases} \varrho_f(t+1) = \varrho_f(t) + \gamma_\varrho \cdot \dfrac{\partial L}{\partial \varrho_f}(\varrho(t), \lambda(t)) \\ \lambda_l(t+1) = \lambda_l(t) - \gamma_\lambda \cdot \dfrac{\partial L}{\partial \lambda_l}(\varrho(t), \lambda(t)) \end{cases} \tag{3}$$

Expanding the partial differentiates, the previous update equations are rewritten:

$$\begin{cases} \varrho_f(t) = \varrho_f(t) + \gamma_\varrho \cdot \Big(U'_f(\varrho_f(t)) - \displaystyle\sum_{l \text{ used by } f} \lambda_l(t) \Big) \\ \lambda_l(t) = \lambda_l(t) - \gamma_\lambda \cdot \Big(B_l - \displaystyle\sum_{f \text{ through } l} \varrho_f \Big) \end{cases} \tag{4}$$

λ_l is generally called shadow price for link l and the previous equations then lead to a surprisingly simple algorithm that can be interpreted as follows (See Figure 1):

- Every flow f evaluates the “total price” of the resources it uses (i.e., the sum of the $\lambda_l(t)$) and adapts its emission rate to account for both its utility and the virtual price it should pay. Whenever the price gets “too expensive”, the flow decrease its emission rate and conversely.

- Every resource l evaluates whether it is saturated or not and adapts its price accordingly. Whenever a resource is saturated, it will increase its price so that the flows going through it decrease their usage and whenever a resource is underused, it will lower its price so that the flow going through it can increase their rate.

This “offer-and-demand” inspired algorithm is a simultaneous gradient descent on both primal and dual variables that will converge to the saddle point, which

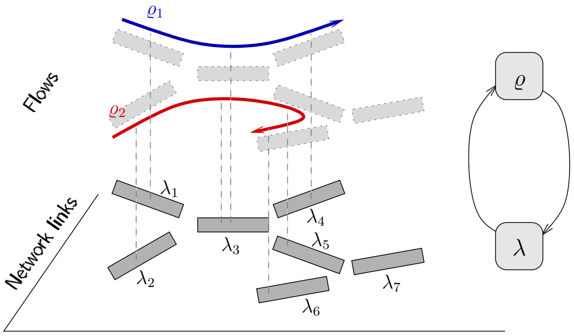


Figure 1: Distributed sharing algorithm based on Lagrangian optimization and gradient descent. Flows adapt their rate ϱ based on prices λ advertized by the network links they use and conversely.

is the optimal solution of the original problem. By adapting the step-size, or the utility functions, one gets different protocol. Such technique have for example been used either to design protocols achieving a given fairness criteria or to reverse-engineer existing protocols. For example, by making an analogy between the window adjusting protocols and the primal update equations, Low *et al.* proved [14] that TCP Vegas achieves some form of proportional fairness, while first versions of TCP Reno behaved as if utility functions were arctan based functions.

To sum up, this approach is based on the following three steps:

1. **Modeling.** Model the problem as a concave non-linear maximization problem;
2. **Partial differentiates.** Convert this problem into two coupled optimization problems using Lagrangian and compute the partial differentiates of L with respect to each primal and dual variables;
3. **Algorithm design.** From the structure of these partial differentiates, devise a distributed algorithm implementing coupled gradient descent and ascent.

The key ingredients to turn the partial differentiates into a distributed algorithms (i.e., move from step 2 to step 3) are (1) the separability of objective function (it is a sum over the flows of quantities that depend only on each flow) and (2) the structure of the constraints (each constraint can be associated with a resource).

2.3 Flow Control in Multi-path Network

A similar approach relying on these three steps has been used in the context of network flows that may choose among a predetermined set of routes [19]. In such a context, each flow f is subdivided into sub-flows $f_1,\ldots,f_k$ and the optimal flow control is written as:

$$\begin{array}{l} \max \sum_{f\in\mathcal{F}} U_f\left(\sum_k \varrho_{f,k}\right) \\ \left\{ \forall l \in \mathcal{L} \displaystyle\sum_{f_k \text{ going through } l} \varrho_{f,k} \leqslant B_l \right. \end{array} \tag{5}$$

Wang *et al.* [19] specifically addressed this problem with the additional constraint that each flow has some minimum and maximum requirements: $\forall f, m_f \leqslant \varrho_f \overset{\text{def}}{=} \sum_k \varrho_{f,k} \leqslant M_f$. As this kind of constraints is not relevant in our context, for the sake of clarity, we only present in the following, simplified versions of the equations and algorithms proposed by [19].

Now that the first *modeling step* is achieved, we can move on to the *partial derivatives step*. Using the same technique as previously, a constant step-size gradient algorithm leads to the following update:

$$\begin{cases} \varrho_{f,k}(t) = \varrho_{f,k}(t) + \gamma_\varrho \cdot \Big(U_f'(\varrho_f(t)) - \sum_{l \text{ used by } f_k} \lambda_l(t)\Big) \\ \lambda_l(t) = \lambda_l(t) - \gamma_\lambda \cdot \Big(B_l - \sum_{f_k \text{ through } l} \varrho_{f,k}\Big) \end{cases} \tag{6}$$

We can now move on to the *algorithm design step*. The main difference with the previous setting is that now each sub-flow f_k has its own rate and that it requires the aggregate throughput of flow f to perform its update. More concretely, each flow f evaluates the price of each sub-flow f_k and updates the sub-flow rates accordingly, slowly moving toward the cheapest alternatives. .

Unfortunately, a technical issue prevents the previous equations to be used bluntly. Since the original objective function is not strictly convex (it is *strictly* convex in any of the ϱ_k, but not with respect to the $\varrho_{f,k}$), the dual function d is not twice differentiable and so, a gradient descent algorithm based on this approach may oscillate and exhibit convergence instabilities. This problem can be circumvented by adding a quadratic term, which makes the primal cost function strictly convex. This technique is called *proximal optimization* (see for example [18], chapter 3) and is used in [19] where two alternative algorithms are proposed to solve the flow control problem in multi-path networks.

Consider the new modified optimization problem:

$$\max_{\widetilde{\varrho}\geqslant 0} \max_{\varrho\geqslant 0} \sum_f U_f\left(\sum_k \varrho_{f,k}\right) - \sum_k \sum_f \frac{c}{2}(\varrho_{f,k} - \widetilde{\varrho}_{f,k})^2, \tag{7}$$

where $\widetilde{\varrho}_{f,k}$ is an auxiliary variable and c a constant (set to 1 in [19]).

At the optimum, $\widetilde{\varrho}_{f,k} = \varrho_{f,k}$ and hence the solution of (7) is the same as the one of (5). This optimization problem is strictly concave in each variable $\widetilde{\varrho}_{f,k}$ and $\varrho_{f,k}$ and is equivalent to

$$\max_{\widetilde{\varrho}\geqslant 0} \max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\widetilde{\varrho}, \varrho, \lambda), \tag{8}$$

where any of the minimization and maximization problem is a convex or concave optimization problem of a twice differentiable function. A classical fixed step-size gradient descent algorithm can be used for each level. Such three-level resolution would however be extremely inefficient in practice as it would require to detect a number of times the convergence (in a distributed way) of the inner problems before further proceeding on the outer problems. This is why Wang *et al.* [19] propose to update all variables ϱ, $\widetilde{\varrho}$ and λ simultaneously, hence breaking the very constraining three-level hierarchical structure of the proximal optimization problem from Eq. (8). In essence, this leads to the same

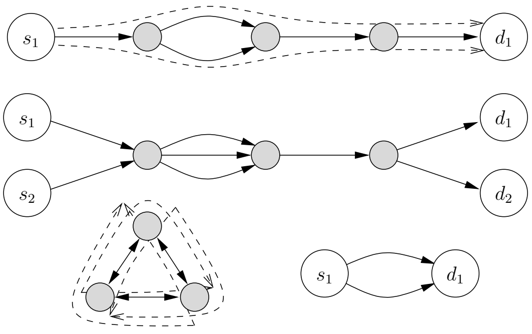


Figure 2: Four test topologies used to illustrate the behavior of Lagrangian based flow control algorithms in multi-path networks. s_i denotes the source of a flow and d_i the destination. Dotted lines are one particular path that a flow may use.

equations and algorithm as (6), except that the $\widetilde{\varrho}$ act as a smoothing term for the ϱ variables to dampen oscillations.

Note that Wang *et al.* do not provide the proof of the convergence of this algorithm. This was studied in a more recent work of Lin and Schroff [20], in a similar setting, where the structure of the two outermost optimization problems is broken. In particular, they prove sufficient conditions on the step-sizes for the algorithm to converge and also study the impact of measurement noise.

More precisely, let us denote by $E_{f,k}$ the routing vector for sub-flow f_k. This means that $E^l_{f,k}$ is equal to 1 if route f goes through link l and 0 otherwise. One of the main result of [20] is that the algorithm converges if the step-size γ_λ satisfies

$$\gamma_\lambda < \frac{c}{2 \cdot S \cdot M}, \text{ where } \begin{cases} M = \max_{f,k} \sum_l E^l_{f,k} \text{ and} \\ S = \max_l \sum_k \sum_f E^l_{f,k} \end{cases} \tag{9}$$

More concretely, M is the length of the largest path and S is the largest number of sub-flows going through a link. Although this problem has been extensively studied on a theoretical point of view, it is interesting to note that, to the best of our knowledge, experimental validation of the resulting algorithms is rather limited. The only tested situations reported in [19, 20] are shown on Figure 2 and involve at most 8 nodes and 3 pairs of sources/destinations. In both studies, the proposed step-sizes for each setting lead to a satisfactory convergence within a few dozens to a few hundreds of iterations.

3 Steady-State Scheduling of BoT Applications: Framework and Models

3.1 Platform Model

Throughout this article, we represent the target computing and communication resources by a *platform graph*, i.e., a node-weighted edge-weighted graph $G = (N, E, W, B)$, as illustrated in Figure 3. Each node $P_n \in N$ represents a computing resource that can deliver W_n floating-point operations per second. Each edge $e_{i,j} : (P_i \to P_j) \in E$ is labeled by a value $B_{i,j}$ which represents the

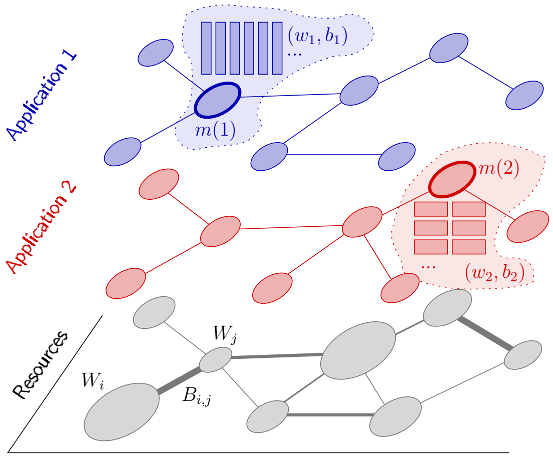


Figure 3: A resource graph labeled with node (computation) and edge (communication) weights. Two application deployments with different sources and task characteristics.

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. For the sake of clarity, we ignore processor-task affinities; instead, we assume that only the number of floating-point operations per second (W_u for processor P_u) determines the application execution speed. However, taking such affinities into account does not change any of the results presented in this article[1].

We assume that all W_i are non-negative rational numbers. For any i, $W_i = 0$ means that P_i has no computing power but yet, can 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). Note that this framework also comprises the bounded multi-port extension [21] 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.

3.2 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 (see Figure 3). Each application is composed of a very large set of independent, equal-sized tasks. We can think of each A_k as a bag of tasks,

[1] It simply amounts to replace the latter-to-be-defined application-specific quantities w_k by $w_{n,k}$.

where the tasks are files that require some processing. A task of application A_k is called a task of *type* k and is described by a computational cost w_k (a number floating point operations) and a communication cost b_k (in bytes) for the associated files. For sake of simplicity, 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. Considering inward data would incur only minor modifications to the remaining equations and algorithms. We further 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 [22]. Therefore, if an application k uses 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. We do not assume that there is a single way to go from a location to another (which generally does not hold true in a grid environment). We rather assume that if several ways exist, only one is actually used.

3.3 Steady-State Scheduling and Fairness

As each application has an unlimited supply of tasks, it should aim at maximizing its average number of tasks processed per time-unit (the ***throughput***). 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 [2] 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.

We denote by ϱ_k the throughput of application k at the steady state: $\varrho_k = \sum_{n \in N} \varrho_{n,k}$. In this article we focus on proportional fairness (i.e., $U_k = \log$) as this measure is *scale-free*[2] but the algorithms we present can be straightforwardly adapted to α-fairness, which accounts for other types of fairness. Therefore, we aim at finding $(\varrho_{n,k})_{1 \leqslant k \leqslant K, 1 \leqslant n \leqslant N}$ that solve

$$\max \textstyle\sum_k U_k(\varrho_k) \\ \begin{cases} \text{(10a)} \quad \varrho_k = \displaystyle\sum_n \varrho_{n,k} \qquad \text{(10b)} \quad \forall n, \displaystyle\sum_k \varrho_{n,k} w_k \leqslant W_n \\ \text{(10c)} \quad \forall (P_i \to P_j), \displaystyle\sum_k \quad \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \leqslant B_{i,j} \end{cases} \tag{10}$$

The first equation (10a) is only introduced for ease of notations. Constraint (10b) states that the computation capacity of processor n cannot be exceeded. Similarly, constraint (10c) states that the network capacity of link $(P_i \to P_j)$ cannot be exceeded.

This framework is very general as it does rely neither on the assumption 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).

[2] It is insensitive to the units in which throughput is expressed. If an application were to group into tasks twice as big, the same resource share would result in a twice smaller throughput. Such scale-free property is highly desirable in our context since throughput is expressed in tasks of application k per time-unit.

4 Decentralized Scheduling of BoT Applications

The optimal BoT application scheduling on grid, as described in Section 3 is very similar to the optimal flow control problem in multi-path routing presented in Section 2.3. Hence, we can apply the same Lagrangian based technique.

Computing Partial Derivatives Applying the Lagrangian methodology to this context leads to the introduction of dual variables for both computing resources (λ_i for P_i) and communication resources ($\mu_{i,j}$ for $(P_i \to P_j)$). Again, the resulting algorithm will be governed by the following dynamic:

$$\begin{cases} \varrho_{n,k}(t+1) = \varrho_{n,k}(t) + \gamma_\varrho \cdot \dfrac{\partial L}{\partial \varrho_{n,k}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \widetilde{\varrho}_{n,k}(t+1) = \widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}} \cdot \dfrac{\partial L}{\partial \widetilde{\varrho}_{n,k}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \lambda_i(t+1) = \lambda_i(t) - \gamma_\lambda \cdot \dfrac{\partial L}{\partial \lambda_i}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \mu_{i,j}(t+1) = \mu_{i,j} - \gamma_\lambda \cdot \dfrac{\partial L}{\partial \mu_{i,j}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \end{cases} \tag{11}$$

Expanding the partial derivatives for $\varrho_{n,k}$, we get:

$$\frac{\partial L}{\partial \varrho_{n,k}} = U_k'(\varrho_k(t)) - \underbrace{\Big(b_k \cdot \sum_{\substack{(P_i \to P_j) \text{ from} \\ \text{m(k) to } P_n}} \mu_{i,j}(t) + w_k \cdot \lambda_n(t) \Big)}_{p_k^n(t)\text{: aggregate price to use } P_n} \tag{12}$$

As expected, the aggregate price to use P_n accounts for both communication link usage (the $\mu_{i,j}$) and CPU usage (the λ_n). Furthermore, this usage is weighted by communication (b_k) and computation (w_k) requirements of application k. Again, updating $\varrho_{n,k}$ requires the knowledge of ϱ_k, which is the aggregate throughput of application k.

Expanding partial derivatives with respect to other variables is straightforward and does not bring particular insight so we do not detail them here. They lead to update equations analogous to Equation (6): as soon as the capacity of a resource is exceeded, its price increases and vice-versa.

Distributed Algorithm Design In grids, the master-worker 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 very few "sources.", which may have some important impact on the convergence rate.

Last, 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 worker) to adjust the rates, so as to prevent overloading the master. Hence the prototype "source" algorithm we proposed in [23] was based on a decentralized aggregation of various quantities, which we detail here.

Using this particular structure, we propose to implement this dynamic using classical traversal algorithms [24] initiated by the master of each application:

- Each node P_n is responsible for primal variables $\varrho_{n,k}$ and $\widetilde{\varrho}_{n,k}$, while the master nodes are responsible for the aggregation ϱ_k of the $\varrho_{n,k}$. Each resource (CPU or link) is responsible for its dual variable (λ_n or $\mu_{i,j}$).
- Each root propagates its aggregate throughput ϱ_k along the tree. During the propagation, the aggregate price for sending data from the master is computed. Therefore, upon reception, each node has all required information to update its contribution $\varrho_{n,k}$ to application k. Upon reception, the leaves of the tree send back their $\varrho_{n,k}$ up-tree, which are in turn aggregated up to the master.

Similarly to the original algorithm of [19], there is no need for any global information, such as the number or the kind of nodes that are in the grid. Nodes only need to communicate with their neighbors and to update the few variables they are responsible for. The wave algorithms seamlessly aggregate all required quantities with no direct interaction among the different applications.

Furthermore, the resulting algorithm only requires very simple computations and small communications. Last, this algorithm seamlessly adapts to variations of W_i of $B_{i,j}$ and to the arrival or departure of new nodes and applications.

Convergence Issues The previous algorithm converges to the optimal solution when provided with an adequate choice of γ_λ,γ_μ,γ_ϱ, and $\gamma_{\widetilde{\varrho}}$, a recurrent problem in gradient based algorithms. Even though the results by Lin and Schroff [20] do not provide any insight on the convergence speed, condition (9) implies that the step-size should be much smaller in the steady-state scheduling context than in the multi-path flow-control problem. Indeed, in the flow-control problem, the vector E is a $0-1$ vector whereas in our context, its values are the w_k and the b_k, that are much larger. The step-size should be at least inversely proportional to $M \cdot L$, where

$$L = \max_k \left(b_k . \left(\max_n |(P_{m(k)} \rightsquigarrow P_n)|\right) + w_k\right) \text{ and}$$

$$M = \max\left(\sum_k w_k, \max_{(P_i \to P_j)} \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} b_k\right).$$

Now, although M and S have less obvious interpretation, one sees that they account for b_k and w_k and that step sizes should thus be significantly small, which may impact convergence speed.

As we previously mentioned, although very promising, experimental studies of these algorithms reported in the literature are rather limited. Our initial evaluation in [23], although limited revealed that even in very simple settings, finding satisfactory step-sizes seemed sometimes impossible and that non trivial adaptation of Equation (12) were required. This is why we devote the rest of this article to demonstrate that application heterogeneity is the source of the difficulty, to explain how step-sizes should be tuned and how updates should be performed to be effective in a wide variety of scenarios.

5 Performance Evaluation of the Naive Algorithm

In this Section, we evaluate the algorithm proposed in Section 4 when using exactly Equation (11). We call such algorithm the “naive algorithm” as it is a

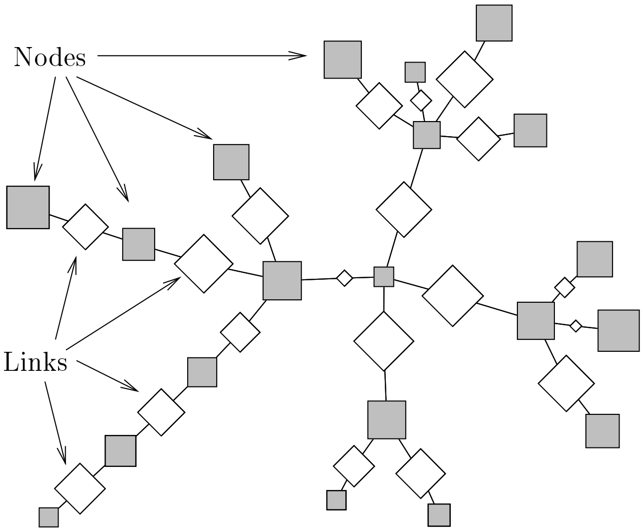


Figure 4: Sample platform with $N = 20$ nodes and maximum degree $d_{max} = 5$. The area of squares (resp. diamonds) is proportional to the capacity of the nodes (resp. links).

straightforward application of [19]. We evaluate our algorithm using the SimGrid simulation toolkit [25]. All our codes, scripts and experiment results are available at http://mescal.imag.fr/membres/arnaud.legrand/distla_2012/.

5.1 Platform Generation

The platforms used for the experiments are random trees described by two parameters: the number of nodes n and the maximum degree of each node d_{max}. The interconnection network topologies are generated using a breadth-first algorithm in order to generate wider (rather than deep and narrow) trees. Figure 4 shows a sample platform with 20 nodes and a maximum node degree of 5. Computational speeds are uniformly chosen in the range $[2, 10]GFLOP/s$, while link capacity is chosen in the range 110 kB/s – 7 MB/s.

5.2 Applications

To evaluate the applicability of our distributed scheduling algorithm, we use three different types of applications, to which the algorithm has to allocate resources efficiently and fairly. As stated above, we carefully designed the algorithm to cope with network-bound applications. Thus, it is important to define application types that exhibit different CCRs. The following three types of applications have been used for the experiments:

A CPU-bound application Each task performs a multiplication of two squared matrices of size $t = 3500$. The size of the tasks are hence $b_1 = 8{\times}2{\times}t^2 =$ 196 MB. The amount of work required is roughly $w_1 = t^3 =$ 42,875 MFLOP. Then, its communication to computation ratio (CCR) is $c_1 = b_1/w_1 = 4.57 \cdot 10^{-3}$.

A network-bound application The tasks of that application implement the addition of two squared matrices of size t. Therefore, the size of each task is $b_2 = 196\,\mathrm{MB}$ and their processing requirements are $w_2 = t^2 = 12.25\,\mathrm{MFLOP}$. This results in a relatively large CCR of $c_2 = b_2/w_2 = 16$.

An intermediate application A task of this application sorts a vector of $t' = 1{,}000{,}000$ double elements (8 bytes). Hence, all tasks are of size $b_3 = 8\,\mathrm{MB}$ and their processing requirements are $w_3 = t' \cdot \log(t') = 13.81\,\mathrm{MFLOP}$. The CCR of this application is $c_3 = 0.58$.

Note that the CCR of such applications is much larger than for those which are typically encountered in BOINC projects [3]. They are therefore more difficult to schedule. For the experiments, the master of each application is chosen randomly, but no two applications are emitted from the same host. Although the previous computation of b and w is very crude and quite unrealistic for the aforementioned applications, it enables to span a rather difficult set of scenario.

5.3 Validating Results

Determining convergence of iterative algorithms is generally not easy. Fortunately, it has been shown that the problem of finding a fair allocation of resources subject to linear constraints can be expressed in an SDP (Semi-Definite Programming) program [26]. Therefore, in the experiments, we test the convergence of our algorithm by comparing the computed objective value to the response of the DSDP solver [27]. Since the SDP program can be solved in polynomial time, it provides a quick and reliable, yet centralized, validation of our numerical results.

5.4 Convergence

In the following, we consider convergence in terms of value of the objective function, as it represents the fairness and efficiency of the solution. Suppose that at time epoch t, the objective value is at 95% of the optimal. Then, this means that the allocation is Pareto efficient (one cannot increase the throughput of an application without needing to decrease another one), and fair (as values of the throughput are relatively close to each others, in accordance with the measure of the objective function). Yet, while the points may be close to the optimal ones, in terms of the objective function, they may be arbitrarily far away in the original ones (i.e., in the $\varrho_{n,k}$ space).

As described earlier, the convergence of our algorithm is determined by comparing the current objective value of our algorithm to the value of the SDP solver. Since our objective function is a sum of logarithms, we consider the solution to have converged with a precision of $0 < x \leqslant 1$ if the objective value lies within the interval $[obj_{opt} + \log(x), obj_{opt} - \log(x)]$, where obj_{opt} is the optimal solution obtained by SDP. However, while performing the gradient descent, oscillations may occur. So, the objective value may lie within the correct interval in iteration v but not in iteration $v + 1$. For that reason, we run the algorithm for a maximum number of iterations v_{max} and check if the objective value of our algorithm is within the computed bounds around the optimal SDP value for the last v_{check} iterations. For all the experiments, we use a precision of $x = 0.85$

(85%) and we consider the algorithm to have converged if the objective value remains within bounds for the last $v_{check} = 100$ iterations.

5.5 Results for Homogeneous Applications

To assess the convergence of our algorithm, we start with experiments using an homogeneous (in term of CCR) set of three applications (three intermediate applications, i.e., $w = 13.81 \times 10^6$ and $b = 8 \times 10^6$). For each experiment, we randomly select the master node of each application on a given platform.

To assess the convergence of our algorithm, we have started with experiments using three homogeneous applications in terms of CCR. For each experiment, we randomly selected the master node of each application on a given platform. We have used a set of three intermediate applications ($w = 13.81 \times 10^6$ and $b = 8 \times 10^6$) for conducting the experiments.

5.5.1 Small platforms

For this first experiment, we use three different platform sizes: 5 nodes, 10 nodes, and 20 nodes. Since the shape of the platform depends on the degree of each node, we vary the degree for these platforms and generate 9 random platforms: three for $N = 5$ nodes and maximum degree $d_{max} = 3$; three for $N = 10$ with $d_{max} \in \{3, 5, 7\}$; and three for $N = 20$ with $d_{max} \in \{5, 10, 20\}$. Through an extensive search, we find a set of parameters for which our distributed algorithm converges for the given homogeneous applications on all experimental platforms. These parameters are: $\gamma_\varrho = 0.01$, $\gamma_{\widetilde{\varrho}} = 0.1$, $\gamma_\lambda = 1 \times 10^{-14}$, $\gamma_\mu = 1 \times 10^{-14}$. For this set of parameters our algorithm converges within the first 200 iterations, i.e., the objective value enters the tube centered around the SDP value after at most 200 iterations and remains in this tube until iteration 1,500 where we stop the simulation.

These results confirm that the algorithm converges for homogeneous applications, similar to the simulations conducted by Wang *et al.* [19]. Nonetheless, we have seen in this experiment that the convergence is rather sensitive to the chosen parameters. Exhaustive search is rather tedious and is not likely to work when exploring larger sets of platforms. Furthermore, such an approach does not give any statistical information about the sensibility to platform configuration or algorithm parameters. For that reason, we rely on factorial designs for the experiments on bigger platforms [28].

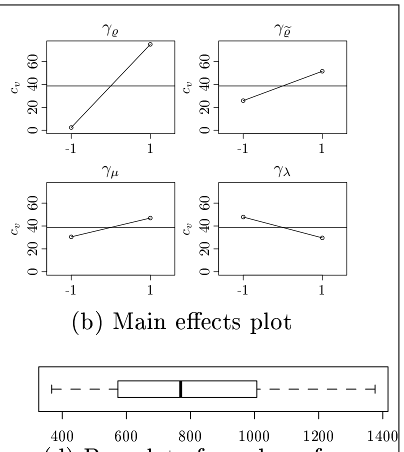
5.5.2 Coefficient of variation

Considering only the final convergence as metric for the experiments leads to only one categorical variable of binary value “converged” or “not converged”. Statistical analysis of the results, e.g., by using an analysis of variance (ANOVA) can be misleading for categorical data [29]. Furthermore, stating whether the algorithm has already converged depends on the computation of the optimal value with the SDP solver that exhibited severe scalability and numerical stability issues in the experiments we conducted[3]. In order to be less dependent of the SDP value and still gain insights on how well the gradient descent works, we

[3] Note that this is problablyy not due to the solver in itself but rather to the encoding into an SDP program that was directly inspired from [26] and may have been better conditionned.

	γ_{ϱ}	$\gamma_{\widetilde{\varrho}}$	γ_{μ}	γ_{λ}
low (-1)	0.001	0.001	1.00e-15	1.00e-15
high (1)	0.1	0.1	1.00e-13	1.00e-13

(a) Parameters for factorial design



(b) Main effects plot

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
ϱ	1	634854.4356	634854.4356	75.1771	0.0000	***
$\widetilde{\varrho}$	1	79293.1822	79293.1822	9.3896	0.0023	**
λ	1	39882.0712	39882.0712	4.7227	0.0303	*
μ	1	32497.4344	32497.4344	3.8482	0.0504	.
platform	29	470172.3060	16212.8381	1.9199	0.0032	**
$\varrho : \widetilde{\varrho}$	1	67441.2012	67441.2012	7.9861	0.0049	**
$\varrho : \lambda$	1	27584.2533	27584.2533	3.2664	0.0714	.
$\widetilde{\varrho} : \lambda$	1	330.0890	330.0890	0.0391	0.8434	
$\varrho : \mu$	1	38450.8702	38450.8702	4.5532	0.0334	*
$\widetilde{\varrho} : \mu$	1	31651.5862	31651.5862	3.7481	0.0535	.
$\lambda : \mu$	1	14136.1931	14136.1931	1.6740	0.1964	
$\varrho : \widetilde{\varrho} : \lambda$	1	1489.7248	1489.7248	0.1764	0.6747	
$\varrho : \widetilde{\varrho} : \mu$	1	35856.2101	35856.2101	4.2460	0.0399	*
$\varrho : \lambda : \mu$	1	16345.9826	16345.9826	1.9356	0.1649	
$\widetilde{\varrho} : \lambda : \mu$	1	16.1140	16.1140	0.0019	0.9652	
$\varrho : \widetilde{\varrho} : \lambda : \mu$	1	158.4895	158.4895	0.0188	0.8911	
Residuals	435	3673480.0717	8444.7818			

(c) ANOVA results: * means that parameter is significant

(d) Box plot of number of iterations until convergence for best parameter set

Figure 5: Experimental results for homogeneous applications with platforms of $N = 20$ and $d_{max} = 5$.

Table 1: Overview of factor levels for factorial experiments with the homogeneous applications.

	N	d_{max}	γ_{ϱ}		$\gamma_{\widetilde{\varrho}}$		γ_{μ}		γ_{λ}		
#platforms	nodes	degree	low	high	low	high	low	high	low	high	nb converged
30	20	5	**0.001**	0.1	**0.001**	0.1	**1E-15**	1E-13	1E-15	**1E-13**	28 / 30
30	20	15	**0.001**	0.1	0.001	**0.1**	**1E-15**	1E-13	1E-15	**1E-13**	29 / 30
30	40	5	**0.001**	0.1	**0.001**	0.1	1E-15	**1E-13**	1E-15	**1E-13**	27 / 30
30	100	5	**0.001**	0.1	**0.001**	0.1	1E-15	**1E-13**	1E-15	**1E-13**	24 / 30

use the coefficient of variation c_v ($c_v = SD(z)/MEAN(z)$, where z is the vector of objective values of the last v iterations) to assess whether the objective value is oscillating or not.

It is important to note that a small value of c_v does not mean that the algorithm has already reached the optimal value. It may be far from the optimal solution but converge very slowly. Generally speaking, small values of c_v and a non-optimal solution reflects that step sizes are too small to let the algorithm converge within the defined maximum number of iterations.

5.5.3 Large platforms

Since we already have a set of parameters that works for small platforms, we base the initial range of values for our factorial design on these values. Figure 5 summarizes the results of the factorial experiment using 30 platforms with $N = 20$ nodes and $d_{max} = 5$.

This summary becomes very useful when seeking a good set of parameters for a given platform size. The ANOVA is computed using a linear combination of all factors while the coefficient of variation c_v is used as response variable. The term “factor ϱ” (resp. $\widetilde{\varrho}, \lambda, \mu$) is used instead of “factor γ_{ϱ}” (resp. $\gamma_{\widetilde{\varrho}}, \gamma_{\lambda}, \gamma_{\mu}$) when there is no ambiguity. The ANOVA table reveals which factor has a real effect on the response. In Figure 5, the factor ϱ is the most influencing factor with a high significance (p value is smaller than 0.0001). This effect can also be seen in Figure 5(b) as the average c_v value is much lower if ϱ is set to its *low* value (-1). This information can be used to select good step sizes for each variable. Yet, one must be careful about the significance results for each factor as well as about possible interactions between parameters. In the present experiment, one could try to reduce further the step size of ϱ since it is the most significant factor. It may however negatively affect the convergence rate. λ and μ have limited impact and can thus *a priori* be arbitrarily chosen in the range $[1 \times 10^{-13}, 1 \times 10^{-15}]$. Note that it can be very hard to draw such conclusions in presence of strong interactions between variables and that further investigations would be required.

By using the parameter set $\gamma_\varrho = 0.001$, $\gamma_{\widetilde{\varrho}} = 0.001$, $\gamma_\lambda = 1 \times 10^{-13}$, $\gamma_\mu = 1 \times 10^{-15}$, the algorithm converges on 28 platforms out of 30 and the corresponding convergence time distribution is given in Figure 5(d). When investigating the remaining two platforms, we observe that 1,500 iterations are simply not enough and the algorithm requires a few extra hundreds of iterations to converge. Further tuning of step sizes could be done using such technique but our goal here is mainly to illustrate how factorial design and ANOVA can help to quickly get some sound analysis of the effects of our algorithm's parameters on convergence. It is important to recall that this approach is not about finding the best possible step sizes for a particular platform but rather to find step sizes that are effective in a wide range of settings.

Table 1 shows an overview of values that have been used for conducting the factorial experiments, where each row specifies the factor levels for a particular platform size. The best combination of factor levels we obtain for each platform size is marked in bold, while the last column holds the number of experiments that have converged in less than 1500 iterations for these best factor levels.

In summary, the factorial design of the experiments can help us to find the correct step sizes that enable the algorithm to converge. The levels of each factor, reported in Table 1, did not have to be modified throughout the experiments for different platform sizes even though the best levels for a platform size have changed.

5.6 Results for Heterogeneous Applications

Since the naive algorithm has proved to be effective in the case of homogeneous applications for a wide range of platforms, we now evaluate the case of heterogeneous applications. In this experiment, three different applications (one instance from each of the three application types that have been defined in Section 5.2) emit tasks to the distributed platform. Again, we use a factorial design for the experiment and start with 30 platforms of 20 nodes and a maximum degree of 5.

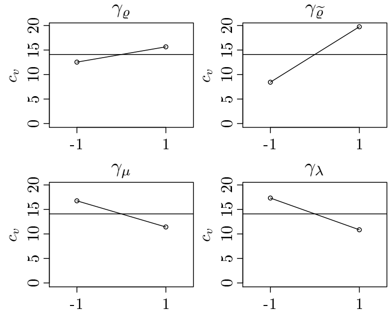
The summary of the experimental results is shown in Figure 6. In this experiment, we use the parameters that worked well in the homogeneous case. However, in contrast to the homogeneous case, the algorithm fails converge in any of the 480 experiments. We can also see that ϱ is the only factor which is not highly significant. It is thus very hard to conclude anything from the ANOVA since there are many significant interactions between all factors. Although, the main effects plot suggests to decrease $\widetilde{\varrho}$ and to increase λ, there is a significant interaction between these two and changing the levels does in such a way may

	γ_{ϱ}	$\gamma_{\widetilde{\varrho}}$	γ_{μ}	γ_{λ}
low (-1)	0.001	0.001	1.00e-15	1.00e-15
high (1)	0.1	0.1	1.00e-13	1.00e-13

(a) Parameters for factorial design

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
ϱ	1	1164.5271	1164.5271	3.7959	0.0520	.
$\widetilde{\varrho}$	1	15367.1284	15367.1284	50.0903	0.0000	***
λ	1	5037.1099	5037.1099	16.4188	0.0001	***
μ	1	3423.1290	3423.1290	11.1579	0.0009	***
platform	29	5547.7536	191.3018	0.6236	0.9387	
$\varrho : \widetilde{\varrho}$	1	3869.2358	3869.2358	12.6121	0.0004	***
$\varrho : \lambda$	1	4546.4969	4546.4969	14.8196	0.0001	***
$\widetilde{\varrho} : \lambda$	1	4277.7955	4277.7955	13.9438	0.0002	***
$\varrho : \mu$	1	3530.3841	3530.3841	11.5075	0.0008	***
$\widetilde{\varrho} : \mu$	1	135.6476	135.6476	0.4422	0.5064	
$\lambda : \mu$	1	31.3388	31.3388	0.1022	0.7494	
$\varrho : \widetilde{\varrho} : \lambda$	1	5972.8812	5972.8812	19.4690	0.0000	***
$\varrho : \widetilde{\varrho} : \mu$	1	5.7724	5.7724	0.0188	0.8910	
$\varrho : \lambda : \mu$	1	2905.8965	2905.8965	9.4720	0.0022	**
$\widetilde{\varrho} : \lambda : \mu$	1	261.5465	261.5465	0.8525	0.3563	
$\varrho : \widetilde{\varrho} : \lambda : \mu$	1	3169.3562	3169.3562	10.3307	0.0014	**
Residuals	435	133453.0395	306.7886			

(c) ANOVA results: * means that parameter is significant



(b) Main effects plot

Figure 6: Experimental results for heterogeneous applications of $N = 20$ nodes and $d_{max} = 5$.

not be effective for decreasing coefficient of variation. Additionally, no factor reduces the c_v to value of less than 7, which was found to be a rather large value for allowing convergence in the homogeneous case.

Nonetheless, we decrease the levels of $\widetilde{\varrho}$, μ and λ according to the resulting main effects and rerun the factorial analysis. The values for ϱ remain identical as this factor was insignificant in the previous ANOVA. As expected, the results of the successive factorial experiment are disappointing. Again, our distributed algorithm fails to converge in any of the 480 experiments. Worse, the average coefficient of variance increased for each factor level. In addition, all factors (except μ) and basically all interactions showed a significant difference in variance in the corresponding ANOVA table. The following conclusions can be drawn from these results:

1. The naive version of the distributed algorithm fails to converge for a heterogeneous set of applications. More precisely, despite our efforts, we cannot find a set of parameters that enables the algorithm to converge for several platforms as seen in the homogeneous case.

2. There might be a set of parameters for which the algorithm converges for a given platform, but this set seems very hard to be determine. Hence, for very simple platforms, we tried several extensive search but always failed to find satisfactory step sizes. Either the algorithm is highly unstable or it is so slow that it fails getting close to the optimal. More precisely, in several case, the objective value is very low, increases very slowly and brutally moves very far away, hence taking a very long time before stabilizing again. The system behaves just as if there was a huge instability zone around the optimal value, which prevents any convergence.

Thus, we show in the next section how the algorithm can be adapted to enable convergence for heterogeneous sets of applications.

6 Recipes for Convergence

As demonstrated in the previous section, an algorithm using naively update equations (11) is ineffective in a heterogeneous application setting. This issue was already identified on a very specific example in [23], but application heterogeneity had not been identified as the explicative factor. Through a detailed analysis of several particular cases, we have been able to identify several sources of instability or of slow convergence. In this section, we detail and justify several modifications that need to be done to update equations (11) to eliminate these issues. Our experience indicates that the combination of all these modifications is required to obtain an efficient algorithm but we have not evaluated their respective impact. Such a study would be interesting but is beyond the scope of this article.

6.1 Avoiding Division by Zero

As explained in Section 3.3, a reasonable choice for U_k is the logarithm function. Yet, when substituting this into the update equations[4] for $\varrho_{n,k}$, we get a term $\frac{1}{\varrho_k}$:

$$\varrho_{n,k}(t+1) \leftarrow \varrho_{n,k}(t) + \gamma_\varrho \left(\frac{1}{\varrho_k(t)} - p_k^n(t) \right),$$

where $p_k^n(t)$ is the aggregate cost that application k should "pay" for using resource n. Unfortunately, although in the optimal solution $\varrho_k > 0$, during convergence, one of the ϱ_k may drop to zero or to very small values. A small value of ϱ leads to huge updates and thus to severe oscillations. As mentioned in [19], it is perfectly valid to normalize this update as follows:

$$\varrho_{n,k}(t+1) \leftarrow \varrho_{n,k}(t) + \gamma_\varrho \left(1 - \varrho_k(t) \cdot p_k^n(t)\right). \tag{13}$$

Such rescaling is very classical and already implemented in the naive algorithm. It is thus completely ineffective in a fully heterogeneous context.

6.2 Fast Convergence of the Primal

As we have previously seen, constant step-size gradient descent on a convex function F is done by repeating the following updates: $x(t+1) \leftarrow x(t) - \gamma \nabla F(x(t))$. It is well known that Newton algorithm has much faster convergence than simple gradient projection algorithm. In *Newton's algorithm*, the updates are as follows: $x(t+1) \leftarrow x(t) - \gamma \left(\nabla^2 F(x(t))\right)^{-1} \cdot \nabla F(x(t))$. Inverting the Hessian matrix $\nabla^2 F(x(t))$ is however very time consuming, which is why *approximate Newton methods* are often used [18]. In such methods, the $\nabla^2 F(x(t))$ matrix is often replaced by a simpler matrix (like its diagonal), whose inversion is straightforward and still has the right order of magnitude. Computing the Hessian matrix for our particular problem leads to a non-invertible matrix because of the non-strict convexity of our initial objective function. Considering only diagonal elements, we get a new scaling that replaces Equation (13):

$$\varrho_{n,k}(t+1) \leftarrow \varrho_{n,k}(t) + \gamma_\varrho \left(1 - \varrho_k(t) \cdot p_k^n(t)\right) \varrho_k(t). \tag{14}$$

[4] We always omit the term in $\widetilde{\varrho}$ for sake of readability.

Unfortunately, again, even with very small step sizes to prevent oscillations, this technique (alone) revealed ineffective to improve convergence in our experiments.

6.3 Stability Condition Around the Equilibrium

Our experiments in [23] and in Section 5.6 showed that this inability to converge was due to a strong instability nearby the equilibrium. Such instability 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.

Therefore, in [23], using a natural stability condition, we have proposed the following scaling on dual variables[5]:

$$\lambda_i(t+1) \leftarrow \lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{n,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k}>0} w_k^2 \varrho_k^2(t)} \tag{15}$$

$$\mu_{i,j}(t+1) \leftarrow \mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^n(t) - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k}>0}} b_k^2 \varrho_k^2(t)} \tag{16}$$

Note that this scaling does not require any additional aggregation as all processors already receive ϱ_k to perform the update of ϱ (Equation (13)). When a node does not participate to any computation or when a link does not convey any data, the denominator is equal to zero and the previous updates are thus not well-defined. We need an update for the case where this situation occurs.

6.4 Avoid Division by Very Small Values and Discontinuities

Something important on which we have not insisted yet is that every variable needs to remain positive. Hence, in any such distributed gradient algorithm, if any update steps leads to a negative variable, the variable is set to 0. This kind of projection is done with the operator $[x(t)+u]^+ = \max(0, x(t)+u)$ and is applied to every variable (both primal and dual). It is typical of such methods but raises several issues in our context. Indeed, it may be the case from an iteration to another that a denominator experiences a very important variation, which may cause a brutal negative step. Whenever many dual (resp. primal) variables suddenly drop to 0, it generally causes a large increase of the primal (resp. dual) variables. This is why these projections need to be smoothed. We used the following smooth projection operator, which revealed extremely efficient:

$$[x(t)+u]^{\alpha+} = \max(\alpha \cdot x(t), x(t)+u), \text{with } 0 < \alpha < 1$$

With such updates, variables never suddenly drop to 0. Instead, variables geometrically decrease to zero, until the corresponding resource is used again. In our experiments, we set α to 1/2.

The final version of the update equations we obtain for our adaptive algorithm is summarized in Figure 7.

[5] Equations in [23] did not take the Newton updates on primal variables and were thus slightly different.

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

$$\widetilde{\varrho}_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}}\varrho_{n,k}(t)\right]^{\alpha+}$$

$$\lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{n,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k}>0} w_k^2 \varrho_k^2(t)}\right]^{\alpha+}$$

$$\mu_{i,j}(t+1) \leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^n(t) - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j)\in(P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k}>0}} b_k^2 \varrho_k^2(t)}\right]^{\alpha+}$$

Figure 7: Update formulas for the adaptive algorithm

7 Performance Evaluation of the Adaptive Algorithm

In this section, we assess the convergence quality of our new iterative algorithm on a wide variety of platforms.

7.1 Platforms and Applications

We select the same platforms as the ones we used for evaluating the homogeneous case, i.e., platforms of size 20, 40, and 100. Additionally, we also include platforms with 500 nodes and a maximum degree of 15. We have tested the same heterogeneous set of applications as for the experiments with the naive version of the algorithm. So, three applications (CPU-bound, network- bound, intermediate) are randomly placed on a host and emit their tasks from this host. We also ensure that applications roots are not too far away from each other and that they actually interfere even when platform size increases.

7.2 Results for Heterogeneous Applications

We started our evaluation with 30 platforms of 20 nodes and a maximum degree $d_{max} = 5$. Since the update formulas differ drastically from the ones use in the naive version , we cannot use the same parameter ranges. Hence, we select one platform and perform an initial scan over a wide range of parameters to find suitable values for each factor. This scan suggests to conduct a factorial experiment with the following values: $\varrho = (0.05, 0.15)$, $\widetilde{\varrho} = (0.05, 0.15)$, $\lambda = (0.7, 1.3)$, $\mu = (0.7, 1.3)$. The first entry of the vector denotes the smaller level of each factor. Performing an ANOVA on the results enables to determine a set of parameters that lead our algorithm to converge on 24 out of 30 platforms. The best values are shown on the first line of Table 2.

Table 2: Good step sizes for different platform characteristics for heterogeneous applications.

nodes	degree	ϱ	$\widetilde{\varrho}$	λ	μ	nb converged
20	5	0.05	0.05	1.3	0.7	24 / 30
20	15	0.01	0.15	0.7	1.3	30 / 30
40	5	0.01	0.05	1.3	0.7	28 / 30
100	5	0.01	0.05	0.7	0.7	27 / 30
500	15	0.002	0.05	0.7	0.7	29 / 30

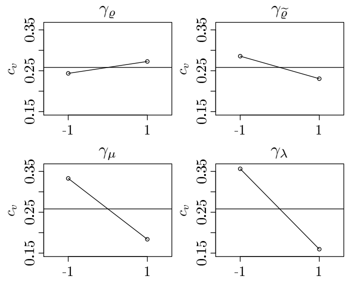
Unfortunately, when testing these factor levels on platforms with 40 nodes and $d_{max} = 5$, these values are found to be ineffective (the algorithm converges

	γ_ϱ	$\gamma_{\widetilde{\varrho}}$	γ_μ	γ_λ
low (-1)	0.001	0.01	0.1	0.1
high (1)	0.002	0.05	0.7	0.7

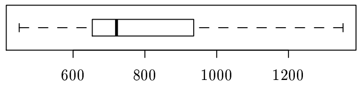
(a) Parameters for factorial design

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
ϱ	1	0.1013	0.1013	2.0243	0.1555	
$\widetilde{\varrho}$	1	0.3640	0.3640	7.2711	0.0073	**
λ	1	4.6328	4.6328	92.5321	0.0000	***
μ	1	2.6296	2.6296	52.5213	0.0000	***
platform	29	28.8514	0.9949	19.8709	0.0000	***
$\varrho : \widetilde{\varrho}$	1	0.0774	0.0774	1.5461	0.2144	
$\varrho : \lambda$	1	0.0796	0.0796	1.5904	0.2079	
$\widetilde{\varrho} : \lambda$	1	0.0410	0.0410	0.8197	0.3658	
$\varrho : \mu$	1	0.0240	0.0240	0.4800	0.4888	
$\widetilde{\varrho} : \mu$	1	0.0045	0.0045	0.0900	0.7643	
$\lambda : \mu$	1	2.3609	2.3609	47.1550	0.0000	***
$\varrho : \widetilde{\varrho} : \lambda$	1	0.1230	0.1230	2.4560	0.1178	
$\varrho : \widetilde{\varrho} : \mu$	1	0.0009	0.0009	0.0182	0.8926	
$\varrho : \lambda : \mu$	1	0.0140	0.0140	0.2798	0.5971	
$\widetilde{\varrho} : \lambda : \mu$	1	0.0071	0.0071	0.1418	0.7067	
$\varrho : \widetilde{\varrho} : \lambda : \mu$	1	0.0020	0.0020	0.0390	0.8436	
Residuals	434	21.7290	0.0501			

(c) ANOVA results: * means that parameter is significant



(b) Main effects plot



(d) Box plot of number of iterations until convergence for best parameter set

Figure 8: Experimental results for heterogeneous applications and the adapted algorithm on platforms of $N = 500$ nodes and $d_{max} = 15$.

in only 7 out of the total 480 experiments). However, the ANOVA reveals that that ϱ is the most significant factor and should be decreased. Thus, we adjust the values for ϱ and run another series of experiments with the following factor levels: $\varrho = (0.01, 0.05)$, $\widetilde{\varrho} = (0.05, 0.15)$, $\lambda = (0.7, 1.3)$, $\mu = (0.7, 1.3)$. The analysis shows that ϱ is again the most significant factor to achieve a small coefficient of variation (c_v) and that the other parameters have little influence. Indeed, when ϱ is set to 0.01, the algorithm converges in 232 of 240 cases.

We conduct further experiments with platforms that are composed of 100 nodes ($d_{max} = 5$) and of 20 nodes ($d_{max} = 15$). We use the same adjusted factor levels that have shown a good convergence quality for 40 nodes. For both experiments, the factor ϱ is the most significant one and needs to be set to its lower value. The best step sizes are also shown in Table 2.

As final experiment, we assess the convergence quality of our algorithm for platforms with 500 nodes and a maximum degree of 15. For these tests, we start again with the factor levels that have been used for platforms of size 40-100, which are $\varrho = (0.01, 0.05)$, $\widetilde{\varrho} = (0.05, 0.15)$, $\lambda = (0.7, 1.3)$, $\mu = (0.7, 1.3)$. When running the experiments, we quickly notice after 40 tests that our algorithm does not converge and that the coefficient of variation is very large. This suggests that our step sizes are too big. Since we have not recorded enough data to conduct an ANOVA, we simply check the distribution of c_v values for the different values of $\varrho = (0.01, 0.05)$. By comparing the histograms, one can see that the c_v values are smaller on average for the smaller value of ϱ. Hence, we decrease the levels of ϱ and conduct the experiments again with these levels: $\varrho = (0.001, 0.01)$, $\widetilde{\varrho} = (0.05, 0.15)$, $\lambda = (0.7, 1.3)$, $\mu = (0.7, 1.3)$. Again, we interactively evaluated the experimental results and stopped the factorial experiment after roughly 170 experiments as many experiments failed to converge. On the data gathered we run an ANOVA and discover that ϱ is again the most significant factor and should be further decreased. Therefore, we lower the levels of ϱ once again to $(0.0001, 0.001)$ and rerun the factorial experiment. Now, the algorithm produces very small values of c_v but again, we never converge within the maximum number of iterations (1500). So, as ϱ was already to set to a small and stable value, we increase the range of ϱ and decrease the range for all other parameters to avoid oscillation. We run a final factorial experiments with the following factor levels: $\varrho = (0.001, 0.002)$, $\widetilde{\varrho} = (0.01, 0.05)$, $\lambda = (0.1, 0.7)$, $\mu = (0.1, 0.7)$. The experimental results are analyzed in Figure 8, which leads us to a set of parameters for which the algorithm converges on 29 out of 30 platforms. Unsurprisingly, the factor ϱ was not significant anymore for obtaining a small c_v as both levels are very close to each other.

Interestingly, the number of iterations required to converge seems to be independent on the size of the platform (as can be observed by comparing Figure 8(d) and Figure 5(d)). Starting arbitrarily far from the optimal solution, the expected number of steps should thus be smaller than 800. A 500 node platform with $d_{max} = 15$ has a diameter of roughly 10 and thus the expected convergence time would be of a dozen of minutes (assuming a $50ms$ RTT between machines). Such convergence time is linear in the number of steps and logarithmic in the platform size. It could certainly be further improved by a better tuning of step sizes and using asynchronous steps.

8 Conclusion

In this article, we have shown the links between communication-bound BoT scheduling in grid platforms with flow control in multi-path networks. Lagrangian optimization and distributed gradient have been extensively used in the latter context and are therefore a very natural technique for the former. Surprisingly, it turns out that although both problems are very similar on a theoretical point of view, the heterogeneity of BoT makes the BoT scheduling problem significantly harder in practice than the flow control problem. Fortunately, we have been able to propose a set of adaptations that lead to an effective fully distributed algorithm for fairly sharing resources between BoT applications. We have evaluated the effectiveness of our algorithm through a set of carefully designed experiments that enable to discriminate real trends from noise introduced by the randomness of the platform. The algorithm is shown to converge in reasonable time even for very large and complex heterogeneous platforms.

Although we propose a (centralized) technique for finding robust step sizes, it turns out that they seem to be dependent on the order of magnitude of the platform characteristics and size (some of them seem to be roughly inversely proportional to platform size) as well as on the order of magnitude of the BoT characteristics and number. The coefficient of variation seems to be a good indicator of convergence and stability and may be used to control step sizes in a real implementation.

References

[1] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, “Steady-State Scheduling on Heterogeneous Clusters: Why and How?” in *Proceedings of*

the 6th Workshop on Advances in Parallel and Distributed Computational Models (APDCM 2004), 2004.

[2] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized Versus Distributed Schedulers Multiple Bag-of-Tasks Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 698–709, May 2008.

[3] "Berkeley Open Infrastructure for Network Computing," http://boinc.berkeley.edu.

[4] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *Proceedings of the 5th IEEE/ACM Intl. Workshop on Grid Computing*, 2004.

[5] 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.

[6] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. A. B. da Silva, C. O. Barros, and C. Silveira, "Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach," in *ICCP*, Oct. 2003.

[7] "Large Hadron Collider Computing Grid," http://lcg.web.cern.ch/LCG/.

[8] H. Casanova and F. Berman, "Parameter Sweeps on the Grid with APST," in *Grid Computing: Making the Global Infrastructure a Reality*, G. F. Fran Berman and T. Hey, Eds. John Wiley & Sons, 2003, ch. 33.

[9] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?" in *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, 2000, pp. 520–528.

[10] M. Litzkow, M. Livny, and M. Mutka, "Condor: A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS'88)*, 1988, pp. 104–111.

[11] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," in *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, 2000, pp. 349–363.

[12] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[13] D. Kondo, A. Chien, and H. Casanova, "Resource Management for Short-Lived Applications on Enterprise Desktop Grids," in *Proceedings of the SuperComputing 2004 (SC'04)*, 2004.

[14] S. Low, "A Duality Model of TCP and Queue Management Algorithms," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 525–536, 2003.

[15] 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.

[16] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

[17] D. P. Bertsekas and R. Gallager., *Data Networks.* Prentice-Hall, 1992.

[18] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, 1989.

[19] 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.

[20] X. Lin and N. B. Shroff, "Utility Maximization for Communication Networks With Multipath Routing," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 766–781, 2006.

[21] B. Hong and V. K. Prasanna, "Adaptive Allocation of Independent Tasks to Maximize Throughput," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1420–1435, Oct. 2007.

[22] 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.

[23] R. Bertin, A. Legrand, and C. Touati, "Toward a Fully Decentralized Algorithm for Multiple Bag-of-tasks Application Scheduling on Grids," in *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*, 2008.

[24] G. Tel, *Introduction to Distributed Algorithms*, 2nd ed. New York, NY, USA: Cambridge University Press, 2001.

[25] A. Legrand, M. Quinson, K. Fujiwara, and H. Casanova, "The SimGrid Project - Simulation and Deployment of Distributed Applications," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, 2006, pp. 385–386.

[26] 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.

[27] S. J. Benson and Y. Ye, "DSDP5: Software For Semidefinite Programming," Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, Tech. Rep. ANL/MCS-P1289-0905, Sep. 2005. [Online]. Available: http://www.mcs.anl.gov/~benson/dsdp

[28] D. C. Montgomery, *Design and Analysis of Experiments.* John Wiley & Sons, Aug. 2005.

[29] T. F. Jaeger, "Categorical Data Analysis: Away from ANOVAs (transformation or not) and towards Logit Mixed Models," *Journal of Memory and Language*, vol. 59, no. 4, pp. 434–446, February 2008.

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

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

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