HAL
open science

Fair scheduling of bag-of-tasks applications using distributed Lagrangian optimization

Rémi Bertin, Sascha Hunold, Arnaud Legrand, Corinne Touati

To cite this version:

Rémi Bertin, Sascha Hunold, Arnaud Legrand, Corinne Touati. Fair scheduling of bag-of-tasks applications using distributed Lagrangian optimization. Journal of Parallel and Distributed Computing, 2013, 10.1016/j.jpdc.2013.08.011 . hal-00872473

HAL Id: hal-00872473

https://inria.hal.science/hal-00872473

Submitted on 13 Oct 2013

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.

Fair Scheduling of Bag-of-Tasks Applications Using Distributed Lagrangian Optimization

Rémi Bertin[a,c], Sascha Hunold[b,c], Arnaud Legrand[b,c], Corinne Touati[a,c]

[a] *INRIA*
[b] *CNRS*
[c] *University of Grenoble. LIG laboratory (MESCAL project), Montbonnot, France*

Abstract

Large scale distributed systems typically comprise hundreds to millions of entities (applications, users, companies, universities) that have only a partial view of resources (computers, communication links). How to fairly and efficiently share such resources between entities in a distributed way has thus become a critical question.

Although not all applications are suitable for execution on large scale distributed computing platform, ideal are the Bag-of-Tasks (BoT) applications. Hence a large fraction of jobs in workloads imposed on Grids is made of sequential applications submitted in the form of BoTs. Up until now, mainly simple mechanisms have been used to ensure a fair sharing of resources among these applications. Although these mechanisms have proved to be efficient for CPU-bound applications, they are known to be ineffective in the presence of network-bound applications.

A possible answer resorts to Lagrangian optimization and distributed gradient descent. Under certain conditions, the resource sharing problem can be formulated as a global optimization problem, which can be solved by a distributed self-stabilizing supply and demand algorithm. In the last decade, this technique has been applied to design network protocols (variants of TCP, multi-path network protocols, wireless network protocols) and even distributed algorithms for smart grids.

In this article, we explain how to use this technique for fairly scheduling concurrent BoT applications with arbitrary communication-to-computation ratio on a Grid. Yet, application heterogeneity raises severe convergence and stability issues that did not appear in previous contexts and need to be addressed by non-trivial modifications. The effectiveness of our proposal is assessed through an extensive set of complex and realistic simulations.

Keywords: Lagrangian optimization, steady-state scheduling, distributed scheduling, grid computing.

1. Introduction

Large scale distributed computing infrastructures are now a reality. Production Grids like EGEE comprise hundreds of sites and several dozens of thousands of processing units.

Email addresses: `sascha.hunold@imag.fr` (Sascha Hunold), `arnaud.legrand@imag.fr` (Arnaud Legrand), `corinne.touati@imag.fr` (Corinne Touati)

Volunteer computing infrastructures such as BOINC comprise over 580,000 hosts that deliver over 2,300 TeraFLOP per day. In such systems, entities have only a partial view of the system, which makes fair and efficient sharing of resources (CPU, network links, storage, ...) among entities (network flows, users, applications, ...) particularly challenging since centralized algorithms do not scale well and distributed algorithms can only rely on local information. A possible approach resorts to the combined use of Lagrangian optimization and distributed gradient descent, which leads to distributed self-stabilizing supply and demand algorithms. In the last decade, this technique, which we call DLO (Distributed Lagrangian Optimization) for short in the sequel, has been applied to design network protocols (variants of TCP [1], multi-path network protocols [2], wireless network protocols [3]) and even distributed algorithms for smart grids [4]. DLO is very appealing as it allows the choice of a wide variety of fairness criteria and achieves both optimal path selection and flow control.

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 among these applications. Although these mechanisms have proved to be efficient for CPU-bound applications, they are known to be ineffective in the presence of network-bound applications. The similitude between the grid context and the multi-path network context indicates Lagrangian-based algorithms as natural candidates for fairly and efficiently scheduling BoT applications.

We first review in Section 3 the context of BoT scheduling and the limitations of existing approaches. Then, we introduce in Section 3 network protocol engineering techniques based on DLO. DLO has been widely used in the networking community and in particular to propose flow control mechanisms in multi-path networks, although to the best of our knowledge their efficiency has been evaluated only in very limited settings.

We explain in Section 4 the similarities between the flow control problem and the problem of fairly sharing communication and computation resources between multiple BoT applications in a grid environment. We show in Section 5 how DLO can be used to design a hierarchical and distributed algorithm. This algorithm only requires local information at each worker process and at each buffer of the network links.

We demonstrate in Section 6 through a carefully designed set of simulations that applying simply DLO to the grid context is effective if and only if all applications are identical: application heterogeneity raises very complex practical convergence issues. Surprisingly, this issue had been completely overlooked in previous works that rather focused on the lack of strict convexity of the global objective function (, which complicates the proof of the convergence but never appeared as a practical issue in our experiments.)

To address application heterogeneity, we detail in Section 7 a set of non-trivial adaptations that are required to ensure convergence. In Section 8, we prove their effectiveness in a fully heterogeneous setting through an extensive set of simulations. We briefly illustrate in Section 9 the ability of the algorithm to adapt to the departure of critical nodes. We believe that our thorough analysis enables the reader to deeply understand the potential benefits as well as the limitations of DLO in the context of grid computing.

The contributions of this article can be summarized as follows:

- A *robust*, *fair* and *optimal distributed scheduling algorithm for concurrent BoT applications with arbitrary communication-to-computation ratio* on Grids. Popular existing

infrastructures do not offer support for applications with such characteristics. The effectiveness of this algorithm is assessed in a wide variety of complex scenarios.

- Our algorithm is based on DLO but requires a set of non-trivial adaptations compared to more classical approaches that can be found in network protocol engineering literature. We provide an experimental proof that although a naive adaptation of DLO is effective when all applications are identical, it is bound to fail when applications have different characteristics. Hence, heterogeneity makes BoT scheduling on grid computing platforms significantly more complex than flow control in multi-path networks.
- We provide an in-depth understanding of our algorithm and of the convergence issues raised in our context by presenting a general introduction to DLO and a comprehensive survey on how it has been used to design network protocols.

2. Scheduling Bag-of-Tasks on Grid Platforms

Not all applications are suitable for execution on large scale distributed computing platforms but ideal are the Bag-of-Tasks (BoT) applications. Hence a large fraction of jobs in workloads imposed on Grids is made of sequential applications submitted in the form of BoT. Despite their suitability for such platforms, scheduling such applications is complexified by several theoretical and practical aspects among which are platform heterogeneity, management of both data and computations, the presence of several users (and even sometimes virtual organizations), fault tolerance, the difficulty to predict workload characteristics as well as their evolution over time and across users. This has lead to a significant amount of efforts on workload characterization, on the design of BoT management infrastructures and on scheduling theory.

In this article, we seek to design a *fair* and *optimal hierarchical scheduling algorithm* for applications with *arbitrary communication-to-computation ratio*. The key characteristic of our work is the consideration of mixtures of CPU-bound and network-bound applications in a multi-user context where a fair sharing of resources needs to be ensured. The Large Hadron Collider Computing Grid (LCG) [5] 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 applications and their efficient management is still an open problem.

2.1. BoT Scheduling Infrastructures

The most well-known systems specifically tailored for BoT in the Grid context are APST [6], Nimrod/G [7], Condor [8], MyGrid [9], Cigri [10] and Glite Workload Management System [11]. All these infrastructures work on a best-effort basis, with no performance guarantees. At a different scale, BOINC [12] is a centralized scheduler that distributes tasks for participating applications, such as SETI@home, ClimatePrediction.NET, World Community Grid or Einstein@Home. Most of the existing systems are client-server oriented and have proved to be efficient for 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 fairly shares on each client the CPU resource among projects to which the volunteer subscribed [12]. Yet, it has been proved [13] that such a simplistic and local approach leads to resource waste whenever communication links become critical resources.

Most existing workload studies [14, 15, 16] ignore communications as such information can generally not be traced at the batch scheduler level. In most currently deployed infrastructures, the file manager and the batch scheduler work in a best effort mode, trying to prefetch data or to schedule computations near data whenever possible. The interaction between efficient data management and scheduling is still not well understood, especially at large scale and under fairness constraints. Hence, simple and pragmatic strategies are used in practice although this lack of understanding motivates a lot of research work in scheduling theory.

2.2. Scheduling Theory

The classical approach for BoT scheduling is to minimize completion time of a single batch while the main issue is to select resources and manage data. Such problems are generally solved with list scheduling heuristics like min-min, sufferage, or similar variations [17, 18, 19]. Many of these heuristics assume a complete knowledge of computation time. Unfortunately, it is generally impossible to assume that nodes are reliable and deliver constant computing power. Such uncertainties are at the heart of pragmatic selection mechanisms and replication is thus often used to avoid waiting for the last tasks of a BoT [20]. This variability issue can also be circumvented by learning characteristics through sampling when BoTs are sufficiently large. This is for example the approach used in [21] and based on the observation [16] that intra-BoT distribution of execution times often follows a normal distribution, which eases parameter estimation.

Introducing notions of fairness in such large scale distributed platforms is difficult and is still the focus of recent research [22]. Some projects like the OurGrid infrastructure use *tit-for-tat* mechanisms inspired by the BitTorrent bandwidth sharing mechanisms [9], but most systems rely on some form of fair sharing or try to achieve it. In practice, fairness is generally a secondary goal and is managed in rather simplistic ways compared to the diversity of approaches that can be found in the game theory literature [23].

Makespan minimization is an intrinsically difficult problem even in a simple setting. In a practical context with communication management, information uncertainty, and fairness constraints, it becomes intractable. That is why relaxations have been proposed, among which are divisible load scheduling [24] and steady-state scheduling [25, 26, 27]. 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 [28].

To the best of our knowledge, most previous work on steady-state scheduling resort to solving a large linear program in a centralized way. Such approach requires to gather information about the global state of the platform and the applications and then to broadcast the solution of the linear program to participants, which makes it sensitive to faults and workload variations, hence hindering practical implementation. The main

exception we are aware of is the work of [29] that proposes a solution based on flow maximization and inspired by push-relabeling algorithm. However this solution is designed for the case of a single user and does not enable to handle fairness among applications with different characteristics. A distributed solution in the case of several applications was proposed in [30], but for a slightly different communication model (the single-port model) and a particular type of fairness (max-min fairness), which complicates the design of distributed solutions. As a consequence, the distributed heuristics proposed in [30] produce schedules whose quality in term of fairness or efficiency cannot be guaranteed. In this article, we explain how to use distributed Lagrangian optimization to circumvent such issues and propose a fully distributed solution to the fair and steady-state scheduling of multiple BoT applications problem.

3. Related Work on Distributed Lagrangian Optimization

3.1. 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 [31, 32]). This technique has also recently been applied to devise supply and demand algorithms for smart grids [4].

Assume we are given a network made of a set of links $\mathcal{L}$ whose capacity B_l for $l \in \mathcal{L}$ is to be shared among a set of flows $\mathcal{F}$. Let us denote by ϱ_f the bandwidth allotted to flow $f \in \mathcal{F}$. Fairly and efficiently sharing resources among 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 lead to different kinds of fairness [1]. Common choices are $U_f(\varrho_f) = \log(\varrho_f)$ (proportional fairness [32]) or $U_f(\varrho_f) = \varrho_f^{\alpha}/(1-\alpha)$ (α-fairness [1], which covers in particular the cases of proportional fairness for $\alpha \to 1$, max-min fairness [33] when $\alpha \to \infty$, social welfare for $\alpha = 0$). Many other fairness definitions have been proposed and some have even been evaluated in the context of cluster management (e.g., [34]). In this article, we focus solely on fair sharing that can be defined as the optimization of the total utility since such kind of objective is flexible and can account for a wide range of situations.

Let us assume that 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:

$$\begin{array}{l} \text{Maximize} \sum_{f\in\mathcal{F}} U_f(\varrho_f) \\ \text{s.t.} \begin{cases} \forall l \in \mathcal{L}, \displaystyle\sum_{f \text{ going through } l} \varrho_f \leqslant B_l \\ \forall f \in \mathcal{F},\ \varrho_f \geqslant 0 \end{cases} \end{array} \tag{1}$$

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. Fortunately, 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 variables ϱ_f of the initial problem are called primal variables. The Lagrangian function is then defined as:

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

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

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

Indeed, if an allocation ϱ is unfeasible, then one of the constraints is violated and the inner minimization problem (the minimization over λ) is thus solved by setting the corresponding λ_l to $+\infty$. Conversely, if ϱ is a feasible allocation, then the inner minimization problem is solved 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 problem is thus strictly equivalent to our original problem. Under very mild assumptions it can be proven that there is no duality gap [35], i.e., that

$$\underbrace{\max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\varrho,\lambda)}_{\text{Primal problem}} = \underbrace{\min_{\lambda\geqslant 0} \max_{\varrho\geqslant 0} L(\varrho,\lambda)}_{\text{Dual problem}} \stackrel{\text{def}}{=} \min_{\lambda\geqslant 0} d(\lambda)$$

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 then 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 and convex minimization problems can be solved through gradient descent (see for example [35, Chapter 3], on the convergence analysis of descent algorithms), the saddle point is generally obtained by applying 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:

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

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

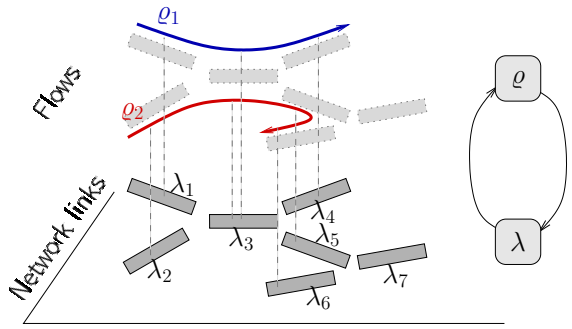


Figure 1: Distributed sharing algorithm based on Lagrangian optimization and gradient descent. Flows adapt their rate ϱ based on prices λ advertised by the network links they use and conversely.

- 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" compared to the utility increase $U'_f(\varrho_f)$, 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 flows going through it can increase their rate.

This "supply and demand" inspired algorithm is a simultaneous gradient descent on both primal and dual variables that will converge to the saddle point, which is the optimal solution of the original problem. By adapting the step-size, or the utility functions, one gets a different protocol. Such techniques 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 [31] that, under some stability assumptions, TCP Vegas achieves some form of proportional fairness, while first versions of TCP Reno behaved as if arctan based utility functions were optimized.

To summarize, the general approach of distributed Lagrangian optimization is based on the following three steps:

1. **Modeling.** Model the problem as a concave non-linear maximization problem;
2. **Partial derivatives.** Convert this problem into two coupled optimization problems using Lagrangian multipliers and differentiate the Lagrangian function L with respect to each primal and dual variables;
3. **Algorithm design.** From the structure of these partial derivatives, devise a distributed algorithm implementing coupled gradient descent (on dual variables) and ascent (on primal variables). This algorithm can be interpreted as a bargaining of applications over resources.

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

3.2. Flow Control in Multi-path Networks

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 [2]. 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:

$$\begin{array}{l} \text{Maximize} \sum_{f\in\mathcal{F}} U_f(\sum_k \varrho_{f,k}) \\ \text{s.t.} \begin{cases} \forall l \in \mathcal{L}, \displaystyle\sum_{f_k \text{ going through } l} \varrho_{f,k} \leqslant B_l \\ \forall f_k \in \mathcal{F},\ \varrho_{f,k} \geqslant 0 \end{cases} \end{array} \tag{5}$$

Wang et al. [2] specifically addressed this problem with the additional constraint that each flow has minimum and maximum requirements: $\forall f, m_f \leqslant \varrho_f \stackrel{\text{def}}{=} \sum_k \varrho_{f,k} \leqslant M_f$. 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 in [2].

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

$$\begin{cases} \varrho_{f,k}(t+1) = \varrho_{f,k}(t) + \gamma_\varrho \cdot \Big(U'_f(\varrho_f(t)) - \displaystyle\sum_{l \text{ used by } f_k} \lambda_l(t)\Big) \\ \lambda_l(t+1) = \lambda_l(t) - \gamma_\lambda \cdot \Big(B_l - \displaystyle\sum_{f_k \text{ through } l} \varrho_{f,k}\Big) \end{cases} \tag{6}$$

We can now move on to the *algorithm design step*. The main difference with the previous setting is that each sub-flow f_k has its own rate and requires the aggregate throughput of flow f to be updated. 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 directly. 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[1]. This technique is called *proximal optimization* (see for example [35, Chapter 3]) and is used in [2] 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 [2]).

[1] Han et al. [36] propose to add a term $\varepsilon \sum_{k,f} \log(\varrho_{f,k}))$ and letting $\varepsilon \to 0$. Although this approach is interesting, it may not be well-suited to a system that needs to operate continuously and where applications and machines regularly join and leave the system.

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 problems 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 several times the convergence (in a distributed way) of the inner problems before further proceeding on the outer problems. This is why Wang et al. [2] 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 equations and algorithm as (6), except that the $\widetilde{\varrho}$ acts as a smoothing term for the ϱ variables to dampen oscillations.

Note that Wang et al. do not provide a proof of the convergence of this algorithm. This was studied in a more recent work of Lin and Schroff [37], 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 effect 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 results of [37] 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 longest path and S is the largest number of sub-flows going through a link.

Although this multi-path flow control 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 [2, 37, 36] involve at most 8 nodes and 3 pairs of sources/destinations. In all theses studies, the proposed step-sizes for each setting lead to a satisfactory convergence within a few dozens to a few hundreds of iterations. Yet, it is difficult to know how sensitive the algorithms are to these step-sizes, as well as how dependent good step-sizes are on the platform shape and size.

Finally, it should be mentioned that beside convexity issues, one of the main problems addressed in most of the previous work is the fact that in network protocols such as TCP or MP-TCP (Multi-Path TCP) [38], the price is interpreted as a measure of congestion and is thus implicitly measured. Such value is hence neither precise nor up-to-date, which creates instabilities. A large part of the existing related work is devoted to solving this problem (see for example [39]). However, as it will be explained in the next sections, such issue does not arise in our context nor in the experiments we performed, as we can develop an application-level protocol where price estimation can follow perfectly the dynamic induced by Lagrangian optimization.

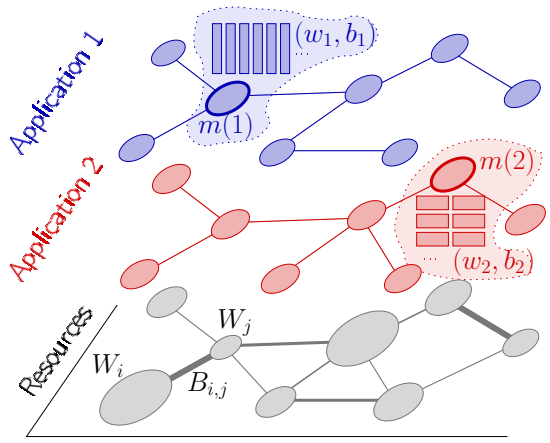


Figure 2: A resource graph labeled with node (computation) and edge (communication) weights. Two application deployments with different sources (respectively $m(1)$ and $m(2)$) and task characteristics.

4. Steady-State Scheduling of BoT Applications

4.1. 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 2. 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 bandwidth between P_i and P_j[2]. 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_n for processor P_n) determines the application execution speed. However, taking such affinities into account does not change any of the results presented in this article.

We assume that all W_i are non-negative rational numbers. If $W_i = 0$, P_i has no computing power but can however 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 its 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 [29] where each node has an additional specific bandwidth bound. This extension would simply amount to change slightly the graph. However, no specific assumption is made on the interconnection graph, which may well include cycles and multiple paths.

[2]Such modeling allows to easily account for asymmetrical connections.

4.2. 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 2). Each application is composed of a very large set of independent, equal-sized tasks. We can think of each A_k as a BoT, 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.

In the sequel to simplify the presentation, 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 [40]. 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 there is no such path 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 only a single way 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. Given the proximity between our problem and the multi-path flow control problem, it would not be difficult to consider several possible routes whenever needed as long as this number of alternatives remains limited. However, it would uselessly complexify the equations and algorithms presented thereafter.

4.3. Steady-State Scheduling and Fairness

As each application comprises 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 [30] 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. Note that the steady-state assumption prevents to account for possible file sharing between tasks, which may be an important aspect but is too irregular to be incorporated in the steady-state scheduling model. When file sharing is a dominant factor and cannot be neglected, other kinds of modeling and techniques should be used (see for example [17, 41, 42]) but to the best of our knowledge, the fairness aspect was never formally considered in such works.

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 context, social welfare optimization (i.e., $\sum_k \varrho_k$) should be avoided as it may lead to starvation of some application. Instead, in this article we choose to focus on proportional fairness (i.e., $U_k = \log$), which is a *scale-free*[3] measure and ensures that no

[3]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.

starvation can occur. Yet, 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

$$\text{Maximize} \sum_k U_k(\varrho_k)$$
$$\text{s.t.} \begin{cases} \text{(10a)} & \varrho_k = \sum_n \varrho_{n,k} \\ \text{(10b)} & \forall n, \sum_k \varrho_{n,k} w_k \leqslant W_n \\ \text{(10c)} & \forall (P_i \to P_j), \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \leqslant B_{i,j} \\ \text{(10d)} & \forall n, \forall k,\ \varrho_{n,k} \geqslant 0 \end{cases} \tag{10}$$

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. If both inward (b_k^{in}) and outward data (b_k^{out})had to be considered, constraint (10c) would be simply replaced by the following constraint:

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

This framework is very general as it neither relies 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).

5. Decentralized Scheduling of BoT Applications

The optimal BoT application scheduling on grids, as described in Section 4 is very similar to the optimal flow control problem in multi-path routing presented in Section 3.2. Hence, we can apply the same Lagrangian based technique.

5.1. Computing Partial Derivatives

Applying the Lagrangian methodology to this context leads to the introduction of dual variables for both computation 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:

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

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

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

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. Note that if both inward and outward data had to be considered, $p_k^n(t)$ would simply comprise an additional term $b_k^{\text{out}} \sum_{(P_i \to P_j) \in (P_n \rightsquigarrow P_{m(k)})} \mu_{i,j}(t)$. Again, updating $\varrho_{n,k}$ requires the knowledge of ϱ_k, which is the aggregate throughput of application k.

5.2. 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, whereas in grids, we would like the intermediate nodes (between a master and each of its workers) to adjust the rates, so as to prevent overloading the master with information management and decision taking. Hence, we propose to use a "source" algorithm[4], which is based on a distributed aggregation of various quantities[5]:

$$\sigma_k^n(t) = \sum_{n' \text{ such that } n \in (P_{m(k)} \rightsquigarrow P_{n'})} \varrho_{n',k}(t) \tag{13}$$

$$\eta_k^n(t) = \sum_{(P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)} \mu_{i,j}(t) \tag{14}$$

σ_k^n is the aggregate throughput of application k at node n. Hence, it reflects how much data will need to flow through node n for application k. η_k^n is the price per byte to pay for sending data from the master $m(k)$ to node n. The price p_k^n can be computed from η_k^n as follows:

$$p_k^n(t) = b_k \cdot \eta_k^n(t) + w_k \cdot \lambda_n(t) \tag{15}$$

[4]This approach was initially proposed in [43] but the update equations showed severe limitations that we explain how to overcome in this article.

[5]Aggregate quantities are noted x_k^n instead of $x_{k,n}$

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

$$\frac{\partial L}{\partial \widetilde{\varrho}_{n,k}} = \varrho_{n,k}(t) - \widetilde{\varrho}_{n,k}(t) \tag{16}$$

$$\frac{\partial L}{\partial \lambda_i} = W_i - \sum_k w_k \cdot \varrho_{i,k}(t) \tag{17}$$

$$\frac{\partial L}{\partial \mu_{i,j}} = B_{i,j} - \sum_k b_k \cdot \sigma_k^j(t) \tag{18}$$

Note that in Eq. (18), one should use either σ_k^j or σ_k^i depending on whether i or j is met first in the deployment of application k.

Using this particular structure, we propose to implement this dynamic using classical traversal algorithms [44] initiated by the master of each application.

Prerequisites Each node P_n is responsible for computing 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 $\mu_{i,j}$). All these variables are initialized with random non-negative values.

Loop Master, workers and resources interact through the following two steps along the application deployment trees in an infinite loop.

- **Step 1: Propagation phase** Each master propagates its aggregate throughput ϱ_k along the tree to the workers. During the propagation, the aggregate price η_k^n for sending data from the master is computed based on the price μ of the communication resources encountered along the path down to node n. Therefore, upon reception, each node has all required information to compute p_k^n and update its contribution $\varrho_{n,k}$ to application k using Eq. (12).
- **Step 2: Aggregation phase** Upon reception of ϱ_k and aggregate communication price, the leaves of the tree send back their new $\varrho_{n,k}$ value up-tree, which are in turn aggregated in σ_k^n up to the master.
  During the aggregation phase, every communication (resp. computation) resource has access to the load $b_k \cdot \sigma_k^j$ (resp. $w_k \cdot \varrho_{n,k}$) incurred by application k and can thus update its price $\mu_{i,j}$ (resp. λ_n).

Such interaction ensures a permanent improvement of resource sharing and a seamless adaptation to variations of W_i of $B_{i,j}$ and to the arrival or departure of new nodes and applications.

Similarly to the original algorithm of [2], 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 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 few message exchanges. A video illustrating the behavior of the algorithm in a simple scenario is available at http://mescal.imag.fr/membres/arnaud.legrand/distla_2012/.

5.3. Convergence Issues

Such kinds of algorithm converge to the optimal solution when provided with an adequate choice of the step-sizes 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 [37] 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 of [37] 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-sizes 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 L have a 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 [43], although limited revealed that even in very simple settings, finding satisfactory step-sizes seemed sometimes impossible and that non-trivial adaptation of Eq. (18) 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.

6. Performance Evaluation of the "Naive" Algorithm

In this section, we evaluate the algorithm proposed in Section 5 when using exactly Eq. (11). We call such algorithm the "naive algorithm" as it is a straightforward application of the distributed Lagrangian optimization proposed in [2] to our context. The only difference lies in the structure of the algorithm. We evaluate our algorithm using the SimGrid simulation toolkit [45] and implementing the whole wave algorithm and synchronization between resources. All our codes, scripts and experiment results are available at http://mescal.imag.fr/membres/arnaud.legrand/distla_2012/.

6.1. 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 to generate wider (rather than deep and narrow) trees. Figure 3 shows a sample platform with 20 nodes and a maximum node degree of 5. Computational speeds are uniformly chosen in the range 2 GFLOP/s – 10 GFLOP/s, while link capacity is chosen in the range 110 MB/s – 7 MB/s.

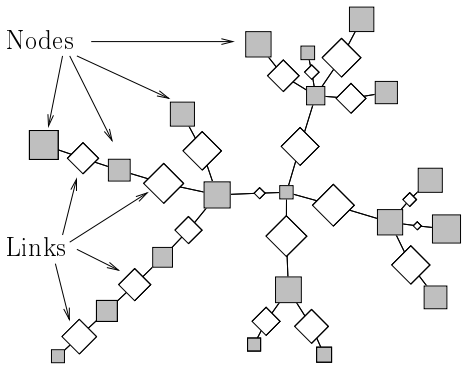


Figure 3: Sample platform with $N = 20$ nodes and maximum degree $d_{max} = 5$. The area of squares (resp. diamonds) is proportional to the capacity of the nodes (resp. links).

6.2. Applications

As we have previously explained in Section 2.1, existing workload characterization studies [14, 15, 16] do not provide any information on communications since such information can generally not be traced at the batch scheduler level. BOINC [12] is clearly a steady-state oriented platform and could be used as a source of inspiration to instantiate our simulations. Yet, applications from BOINC exhibit a particularly low CCR that is not representative of our target use case. Existing studies that try to model BoT characteristics in data grids (e.g., [15, 16]) rely on batch scheduler traces that do not comprise file system information. 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. Three types of applications have been used for the experiments: a CPU-bound application, where each task performs a multiplication of two square matrices of size 3500, hence $b_1 = 196\,\text{MB}$ and $w_1 = 42{,}875\,\text{MFLOP}$; a network-bound application, where the tasks of that application implement the addition of two square matrices of size 3500, hence $b_2 = 196\,\text{MB}$ and $w_2 = 12.25\,\text{MFLOP}$; an intermediate application, where each task sorts a vector of one million double elements, hence $b_3 = 8\,\text{MB}$ and $w_3 = 13.81\,\text{MFLOP}$.

Note that the CCR of such applications is much larger than for those which are typically encountered in BOINC projects. 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. The previous estimation of b and w may seem naive or artificial but they enable to span a rather difficult set of scenarios. Indeed, in practice, the number of applications running in concurrence is rather limited (e.g., the BOINC platform currently comprises around 50 projects for more than 500,000 machines, the LCG runs a dozen of applications on 200,000 processing units). Hence, since the platform sizes we investigate remains rather small, we do not experiment with situations comprising a large number of applications. We also do not investigate how the number of applications influences convergence of the algorithm and instead focus on the platform size and on the nature of applications.

6.3. 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 [46]. Therefore, in the experiments, we test the convergence of our algorithm by comparing the computed objective value to the response of the DSDP solver [47]. Since the SDP program can be solved in polynomial time, it provides a quick and reliable, yet centralized, validation of our numerical results.

6.4. Convergence

The algorithms presented in this article are iterative algorithms whose convergence is only asymptotic. As a consequence, for such algorithms, one always estimates convergence within an arbitrarily fixed precision. 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 relatively efficient (one should not be able to increase the throughput of an application without having 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).

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. Considering tighter thresholds would probably not be really meaningful since real systems experience noise and their characteristics evolve over time.

6.5. Results for Homogeneous Applications

To assess the convergence of our algorithm, we start with experiments using a homogeneous (in terms of CCR) set of three applications (three intermediate applications, i.e., $w = 13.81\,\text{GFLOP}$ and $b = 8\,\text{MB}$). For each experiment, we randomly select the master node of each application on a given platform.

6.5.1. 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 with $N = 5$ nodes and maximum degree $d_{max} = 3$; three with $N = 10$ with $d_{max} \in \{3, 5, 7\}$; and three with $N = 20$ with $d_{max} \in \{5, 10, 20\}$. Through an extensive search, we find a set of parameters (step-sizes) 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. [2]. Nonetheless, we have observed in this experiment that the convergence is rather sensitive to the chosen parameters. Exhaustive search is 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 [48].

6.5.2. 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 [49]. Furthermore, such information is rather poor as it does not convey any information on the reason why the algorithm does not converge. Intuitively, it could be either the case that step-sizes are too large and that the system is unstable or that step-sizes are too small and that the system is evolving very slowly and remains far away from the optimal solution. To gain insights on how well the gradient descent works, we 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 converges 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.

6.5.3. 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 4 summarizes the results of the factorial experiment using 30 platforms with $N = 20$ nodes and $d_{max} = 5$.

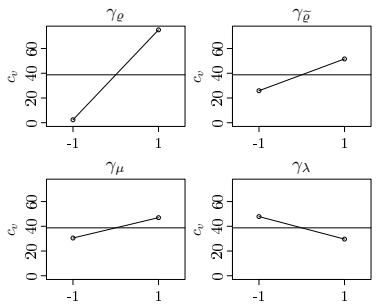
This summary becomes very useful when seeking a good set of parameters for a given platform size. The ANOVA is computed using a linear combination of all factors while the coefficient of variation c_v is used as response variable. The term "factor ϱ" (resp. $\widetilde{\varrho}, \lambda, \mu$) is used 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 4, the factor ϱ is the most influential factor with a high significance (p value is smaller than 0.0001). This effect can also be seen in Figure 4(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

	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

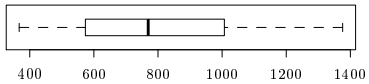
(a) Parameters for factorial design

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
ϱ	1	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



(b) Main effects plot



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

Figure 4: 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		
nodes	degree	low	high	low	high	low	high	low	high	nb converged
20	5	**0.001**	0.1	**0.001**	0.1	**1E-15**	1E-13	1E-15	**1E-13**	28 / 30
20	15	**0.001**	0.1	**0.001**	0.1	**1E-15**	1E-13	1E-15	**1E-13**	29 / 30
40	5	**0.001**	0.1	**0.001**	0.1	1E-15	**1E-13**	1E-15	**1E-13**	27 / 30
100	5	**0.001**	0.1	**0.001**	0.1	1E-15	**1E-13**	1E-15	**1E-13**	24 / 30
500	15	**0.001**	0.1	**0.001**	0.1	**1E-15**	1E-13	1E-15	**1E-13**	20 / 30

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 $\gamma_\varrho = 0.001$, $\gamma_{\widetilde{\varrho}} = 0.001$, $\gamma_\lambda = 1\times10^{-13}$, $\gamma_\mu = 1\times10^{-15}$, the algorithm converges on 28 platforms out of 30 and the corresponding convergence time distribution is given in Figure 4(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 obtained 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. Interestingly, in our experiments, setting $\gamma_{\widetilde{\varrho}}$ to a very small value was better although this parameter had generally little influence. Correctly setting the other step-sizes was however crucial to obtain a robust configuration. Note that one should not believe that convergence is particularly harder to achieve when platform size increases. The fact that only 20 platform out 30 converged within desired time bound for very large platforms is simply the consequence that the step-size values used in this set of experiments are slightly more suited to small platforms than to large ones.

6.6. 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 6.2) 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.

For lack of space, we omit the summary of the experimental results but all details can be found in an extended version published as a research report [50]. In this experiment, we use the parameters that worked well in the homogeneous case. However, in contrast to the homogeneous case, the algorithm fails to converge in any of the 480 experiments. All factors and combinations of factors (but ϱ) are significant and it is thus very hard to conclude anything from the ANOVA. Additionally, no factor combination can reduce the c_v to a value of less than 7, which was found to be a rather large value for allowing convergence in the homogeneous case.

Therefore, we decrease the levels of $\widetilde{\varrho}$, μ and λ 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 experiments are disappointing. Again, our distributed algorithm fails to converge in any of the 480 experiments. Worse, the average coefficient of variation increases 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 determine. Hence, for very simple platforms, we tried several extensive searches 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 cases, the objective value is very low, increases very slowly and suddenly 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.

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

7. Recipes for Convergence

As demonstrated in the previous section, an algorithm using naive update equations (11) is ineffective in a heterogeneous application setting. This issue was already identified on a very specific example in [43], but application heterogeneity had not been identified as the explanatory factor. Through a detailed analysis of several particular cases, we have been able to identify several sources of instability or slow convergence. In this section, we detail and justify several modifications of 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.

7.1. Avoiding Division by Zero

As explained in Section 4.3, a reasonable choice for U_k is the logarithm function. Yet, when substituting this into the update equations 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 (see Eq. (18)). 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 [2], it is perfectly valid to normalize this update as follows:

$$\varrho_{n,k}(t+1) \leftarrow \varrho_{n,k}(t) + \gamma_\varrho \left(1 - \varrho_k(t) \cdot p_k^n(t)\right). \tag{19}$$

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

7.2. 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's 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 [35]. 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 Eq. (19):

$$\varrho_{n,k}(t+1) \leftarrow \varrho_{n,k}(t) + \gamma_\varrho \left(1 - \varrho_k(t) \cdot p_k^n(t)\right) \varrho_k(t). \tag{20}$$

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

7.3. Stability Condition Around the Equilibrium

Our experiments in [43] and in Section 6.6 showed that this inability to converge was due to a strong instability nearby the equilibrium. Every time the objective function goes nearby the optimal value, it instantaneously bounces away. Such instability is caused by 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[6].

Let us assume that we have reached the equilibrium for both primal and dual variables. Further assume that the price λ_n of P_n is increased by $\Delta\lambda_n$. From Equation (20), we derive that such an increase incurs a variation $\Delta\varrho_{n,k}$ of $\varrho_{n,k}$ when $\varrho_{n,k} > 0$:

$$\Delta\varrho_{n,k} = -\gamma_\varrho w_k \varrho_k^2 \cdot \Delta\lambda_n.$$

In turn, from Equation (17), we see that such a variation incurs a variation of λ_n:

$$\sum_{k \text{ s.t. } \varrho_{n,k} > 0} \gamma_\lambda w_k \cdot \Delta\varrho_{n,k} = -\left(\sum_k \gamma_\lambda \gamma_\varrho w_k^2 \varrho_k^2\right) \cdot \Delta\lambda_n.$$

Although the new variation is in reverse direction, the solution of our dynamic will be stable only if the new variation has a smaller amplitude than the initial one, i.e., if for every node n we have:

$$\sum_{k \text{ s.t. } \varrho_{n,k} > 0} \gamma_\lambda \gamma_\varrho w_k^2 \varrho_k^2 < 1.$$

Likewise, we get that the solution of our gradient descent is stable only if for every $(P_i \to P_j)$ we have:

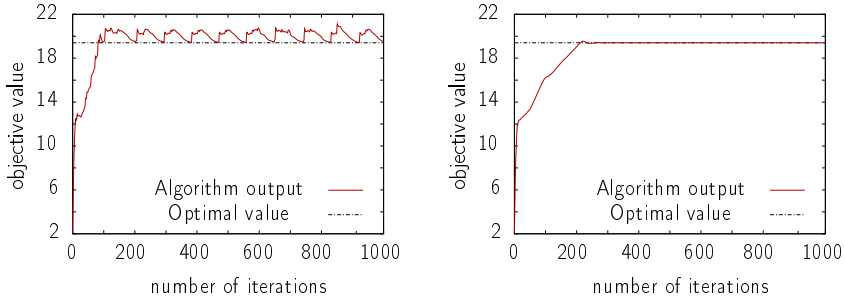
$$\sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k} > 0}} \gamma_\mu \gamma_\varrho b_k^2 \varrho_k^2 < 1.$$

Therefore, the updates of λ and μ should be done accordingly:

$$\lambda_i(t+1) \leftarrow \lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{n,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k} > 0} w_k^2 \varrho_k^2(t)} \tag{21}$$

$$\mu_{i,j}(t+1) \leftarrow \mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^n(t) - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k} > 0}} b_k^2 \varrho_k^2(t)} \tag{22}$$

Note that this scaling does not require any additional aggregation as all processors already receive ϱ_k to perform the update of ϱ (Eq. (19)). When a node does not participate in 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.



(a) Setting the prices to zero if a division by zero occurs

(b) Damping with geometrical update

Figure 5: Stabilization using geometric update.

7.4. Avoid Division by Very Small Values and Discontinuities

An important point on which we have not insisted yet is that every variable needs to remain non-negative (hence, the need in the previous updates of primal variables to only consider variables $\varrho_{n,k}$ that are positive). Hence, in any such distributed gradient algorithm, if any update step leads to a negative value, 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 for 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 large 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 $1/2$. As illustrated in Figure 5, this technique proves very efficient for removing oscillations.

During our investigations, we observed that Newton method on the primal, stability condition on the dual and geometric updates all improved convergence without completely solving the issue. Removing any of these ingredients leads to a dynamic that generally exhibits severe convergence issues, which makes it hard to evaluate their respective influence. Our new adaptive algorithm is thus governed by the following equations:

[6]Such issue had been already reported in [43] but the proposed solution did not take the previous Newton updates on primal variables and were thus slightly different and not as effective.

$$
\begin{aligned}
\varrho_{n,k}(t+1) &\leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\varrho_{n,k}(t) + \gamma_{\widetilde{\varrho}}\widetilde{\varrho}_{n,k}(t) + \gamma_{\varrho}\left(1-\varrho_k(t).p_k^n(t)\right)\varrho_k(t)\right]^{\alpha+} \\
\widetilde{\varrho}_{n,k}(t+1) &\leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}}\varrho_{n,k}(t)\right]^{\alpha+} \\
\lambda_i(t+1) &\leftarrow \left[\lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{i,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k}>0} w_k^2 \varrho_k^2(t)}\right]^{\alpha+} \\
\mu_{i,j}(t+1) &\leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^j(t) - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k}>0}} b_k^2 \varrho_k^2(t)}\right]^{\alpha+}
\end{aligned}
\tag{23}
$$

8. 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.

8.1. 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 application masters are not too far away from each other and that they actually interfere even when the platform size increases.

8.2. 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 used 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 in the first line of Table 2.

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 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.

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

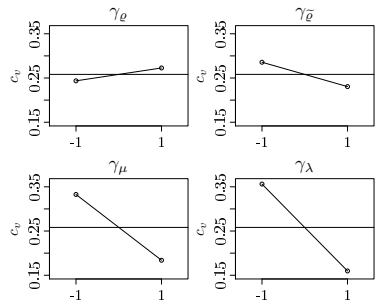
nodes	degree	ϱ	$\widetilde{\varrho}$	λ	μ	nb converged
20	5	0.05	0.05	1.3	0.7	24 / 30
20	15	0.01	0.15	0.7	1.3	30 / 30
40	5	0.01	0.05	1.3	0.7	28 / 30
100	5	0.01	0.05	0.7	0.7	27 / 30
500	15	0.002	0.05	0.7	0.7	29 / 30

	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

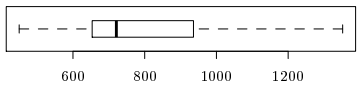
(a) Parameters for factorial design

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

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

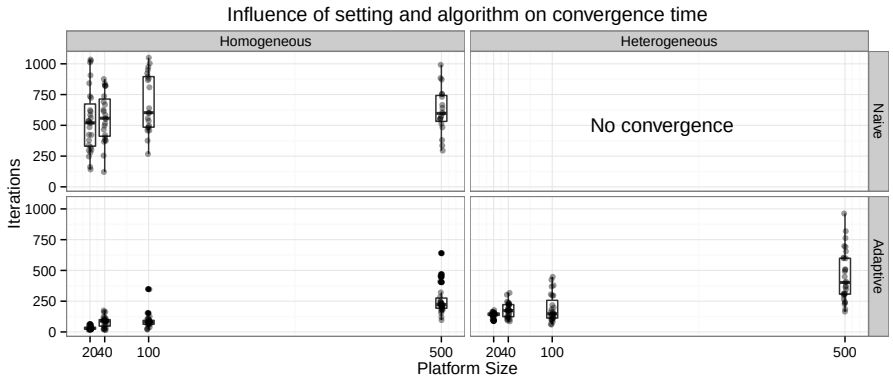


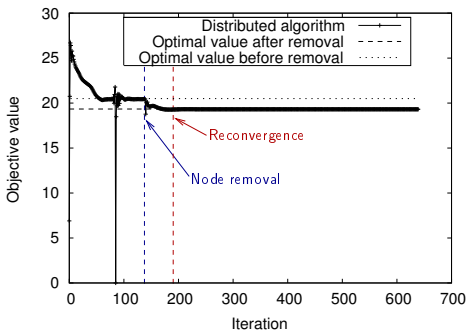
Figure 7: Convergence time distribution as a function of platform size

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 6, 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.

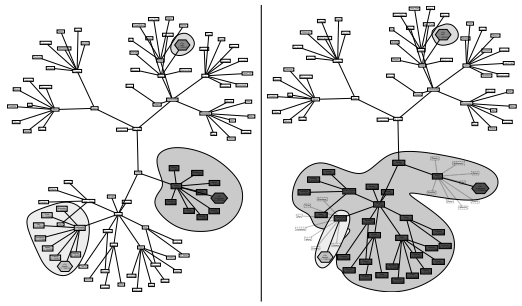
8.3. Convergence Time Analysis

Although the previous procedure is very effective to obtain good step sizes, it turns out that a more careful look at the number of iterations required to converge is highly dependent on platform size. Figure 7 summarizes the evolution of convergence time distribution depending on platform size, application setting (homogeneous vs. heterogeneous) and algorithm (naive vs. adaptive).

At first sight, it may look like convergence time of the naive algorithm in the homogeneous application setting (upper left part of Figure 7) is not really sensitive to platform size. Yet, one should recall that this naive algorithms never converges in the heterogeneous application setting and that only 20 configurations out of 30 converged in the 500 nodes setting whereas 28 configurations out of 30 converged in the 20 nodes setting. Convergence for the 500 node setting could probably be improved as only a crude step-size tuning was done but not to the extent where convergence time dramatically decreases. As we explained in Section 6.6 and as can be seen in the upper right part of Figure 7, the naive algorithm simply never converges when heterogeneous applications are deployed. On the other hand, the adaptive algorithm we proposed converges both in the homogeneous and heterogeneous application setting and even within much better time bounds than the naive algorithm for homogeneous applications.



(a) Worst node removal: all nodes contributing to the computation are suddenly removed from the network at the 140th iteration. The optimum of the objective function decreases, and the algorithm re-converges in a few dozen of iterations, without oscillations.



(b) Platform usage before and after node removal. Hexagons represent the masters and gray nodes participate to the computations. After having removed all nodes involved in computations, one application was redeployed on another part of the network and now interferes with another application.

Figure 8: Behavior of the algorithm when removing all participating nodes.

However, as can be observed on the lower part of Figure 7, platform size has a tremendous impact on convergence time. For a 500 node platform, starting arbitrarily far from the optimal solution, the 95% confidence interval for the expected number of steps is $[373, 531]$. A 500 node platform with $d_{max} = 15$ has a diameter of roughly 10 and thus the expected convergence time would be less than nine minutes (assuming a $50ms$ RTT between machines). It could certainly be further improved by a better tuning of step-sizes and using asynchronous steps.

9. Illustration of Adaptability

As explained in Section 5, as resource shares and prices are constantly reevaluated, such an algorithm is expected to seamlessly adapt to variations of W_i of $B_{i,j}$ and to the arrival or departure of new nodes and applications. Studying such ability is beyond the scope of this article. Yet, we feel that illustrating this ability can help convincing the readers of the interest of such approach. Hence, in this section, we examine the behavior of our algorithm in the case where the system is subject to some perturbations. In particular, we are interested in the removal of nodes, which illustrates the fault tolerance capacity of such algorithm. To this end, we consider a platform of size 100 and show how the system behaves and re-converges when removing *all* the nodes (beside the masters and the nodes connecting them to the network) contributing to a the computations (around 15% of the nodes in this example). Figure 8(a) depicts the evolution of the objective function.

First, one may notice that the optimum of the objective function decreases. This is due to the fact that, as the system is smaller (less nodes are available), the utility set is also smaller. Hence, the objective function being defined on a smaller set has a smaller maximum value than before.

Next, we can see that the objective function only needs around 50 iterations to reach the new optimal equilibrium. As the system had converged to the former value, the initial values (for λ and μ) are relatively close to the new equilibrium. The convergence is thus much faster than when the algorithm is initialized with null values as it was the case in all previous experiments. Note that, in this particular example, the sharing after node removal may seem particularly unfair but it is simply the result of the inability of the light gray application to spread out due to a too high CCR.

10. Conclusion

In this article, we have shown the links between network-bound BoT scheduling in grid platforms and 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 applications makes the BoT scheduling problem significantly harder in practice. As contribution, we have proposed a set of adaptations that lead to an effective fully distributed algorithm for fairly sharing resources between BoT applications. Interestingly, most articles on flow-control for multi-path networks focus on the theoretical difficulty raised by the non-strict convexity of the objective function that hinders the convergence of the resulting algorithm. Yet, in all our experiments, this non-strict convexity has little importance (the step-size of the smoothing factor had always to be set to a very small value and had little influence on convergence) whereas the coupling between primal and dual variable updates and a smooth projection operator are crucial.

All previous work relying on this technique had evaluated the effectiveness of their proposal in very limited settings comprising at most a dozen of nodes and a few pairs of sources/destinations. Instead, we evaluated our algorithms in a much more complex setting with up to 500 node platforms. It is the first time such kind of algorithm is evaluated at such scale and our study reveals issues that had been unnoticed until now. We have evaluated the effectiveness of our algorithm through carefully designed experiments 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, it was not our initial goal, our proposal also improves convergence in the case of homogeneous applications and should thus be effective in the multi-path flow-control setting.

To the best of our knowledge, this is the first guaranteed fully distributed algorithm for steady-state and fair scheduling concurrent BoT applications with arbitrary communication-to-computation ratio on Grids. Although using such algorithm in practice would require further investigation (such as how the number of concurrent applications affect convergence or a thorough study of reconvergence ability), we hope such approach will provide a new perspective to this problem and provide interesting insights on how to design such systems to improve their overall performance.

Although we use a (centralized) technique (designed experiments and ANOVA) 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 could thus be used to control step-sizes in a real implementation.

Some aspects were not explored in our performance evaluation and would deserve further investigation. For example the number of applications and how far the sources of these applications are from each others is likely to play a role in the convergence of the algorithm. Another point that may play a role is inward data when non-symmetric routes are employed as it may generate additional interferences between applications.

Regarding the fundamentals of the algorithm, although the similitude between flow control in multi-path networks and steady-state scheduling of BoT has motivated the use of the augmented Lagrangian method, other methods could be used. As we have seen in Section 7.4, one of the key ingredients for an effective implementation of such method is the smooth projection operator that shares similarities with a barrier function. Lagrangian algorithms move toward the optimal solution by oscillating around the constraints and constantly trying to overuse resources. Besides the resulting potentially slow convergence, the solution may not always be feasible at a given time step. Interior point methods do not suffer from such issues and have received a lots of attention lately. Although they have been used in rather different contexts so far [51], their promising efficiency is a good motivation for trying to adapt them to this setting.

Acknowledgments

The authors warmly thank the reviewers for their insightful comments that allowed to improve this work. This work was supported by the ANR JCJC-07 DOCCA program and by the ANR-08-SEGI-022 USS-SimGrid program of the french government.

[1] J. Mo, J. Walrand, Fair end-to-end window-based congestion control, IEEE/ACM Transactions on Networking 8 (5) (2000) 556–567.

[2] W.-H. Wang, M. Palaniswami, S. Low, Optimal flow control and routing in multi-path networks, Performance Evaluation 52 (2003) 119–132.

[3] J. Jaramillo, R. Srikant, Optimal scheduling for fair resource allocation in ad hoc networks with elastic and inelastic traffic, IEEE/ACM Transaction on Networking 19 (4) (2011) 1125–1136.

[4] Q. Dong, L. Yu, W.-Z. Song, L. Tong, S. Tang, Distributed demand and response algorithm for optimizing social-welfare in smart grid, in: Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium (IPDPS'12), IEEE Computer Society, 2012, pp. 1228–1239.

[5] Large Hadron Collider computing grid, http://lcg.web.cern.ch/LCG/.

[6] H. Casanova, F. Berman, Parameter Sweeps on the Grid with APST, in: G. F. Fran Berman, T. Hey (Eds.), Grid Computing: Making the Global Infrastructure a Reality, John Wiley & Sons, 2003, Ch. 33, pp. 773––787.

[7] D. Abramson, J. Giddy, L. Kotler, High performance parametric modeling with Nimrod/G: Killer application for the global grid?, in: Proceedings of the 14th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2000), 2000, pp. 520–528.

[8] M. Litzkow, M. Livny, 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.

[9] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. A. B. da Silva, C. O. Barros, C. Silveira, Running bag-of-tasks applications on computational grids: The MyGrid approach, in: Proceedings of the 32nd International Conference on Parallel Processing (ICPP 2003), IEEE Computer Society, 2003, pp. 407–416.

[10] Y. Georgiou, N. Capit, O. Richard, Evaluations of the lightweight grid CIGRI upon the Grid5000 platform, in: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (eScience2007), 2007, pp. 279 – 286.

[11] C. Marco, C. Fabio, D. Alvise, G. Antonia, G. Alessio, G. Francesco, M. Alessandro, M. Elisabetta, M. Salvatore, P. Luca, The gLite workload management systems, Journal of Physics: Conference Series 219 (6) (2010) 062039.

[12] D. P. Anderson, BOINC: A system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM Intl. Workshop on Grid Computing (GRID '04), IEEE Computer Society, 2004, pp. 4–10.
[13] A. Legrand, C. Touati, Non-cooperative scheduling of multiple bag-of-task appplications, in: Proceedings of the 25th Conference on Computer Communications (INFOCOM'07), 2007, pp. 427–435.
[14] E. Medernach, Workload analysis of a cluster in a grid environment, in: Proceedings of the 11th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005), Springer, 2005, pp. 36–61.
[15] A. Iosup, M. Jan, O. Sonmez, D. Epema, The characteristics and performance of groups of jobs in grids, in: Proceedings of the 13th International Euro-Par Conference (Euro-Par'07), Springer, 2007, pp. 382–393.
[16] A. Iosup, O. Sonmez, S. Anoep, D. Epema, The performance of bags-of-tasks in large-scale distributed systems, in: Proceedings of the 17th IEEE International Symposium on High Performance Distributed Computing (HPDC-17 2008), ACM, 2008, pp. 97–108.
[17] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), IEEE Computer Society, 2000, pp. 349–363.
[18] 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, 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 61 (6) (2001) 810–837.
[19] Y. C. Lee, A. Zomaya, Practical scheduling of bag-of-tasks applications on grids with dynamic resilience, IEEE Transactions on Computers 56 (6) (2007) 815–825.
[20] D. Kondo, A. Chien, H. Casanova, Resource management for short-lived applications on enterprise desktop grids, in: Proceedings of the ACM/IEEE SC2004 Conference on High Performance Networking and Computing, IEEE Computer Society, 2004, p. 17.
[21] A.-M. Oprescu, T. Kielmann, Bag-of-tasks scheduling under budget constraints, in: Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10), IEEE Computer Society, 2010, pp. 351–359.
[22] P.-F. Dutot, F. Pascual, K. Rzadca, D. Trystram, Approximation Algorithms for the Multi-Organization Scheduling Problem, IEEE Transactions on Parallel and Distributed Systems 99 (11) (2011) 1888–1895.
[23] R. B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, 1997.
[24] B. Veeravalli, D. Ghose, T. G. Robertazzi, Divisible load theory: A new paradigm for load scheduling in distributed systems, Cluster Computing 6 (1) (2003) 7–17.
[25] M. Gallet, L. Marchal, F. Vivien, Efficient scheduling of task graph collections on heterogeneous resources, in: Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'2009), IEEE Computer Society, 2009, pp. 1–11.
[26] H. Casanova, M. Gallet, F. Vivien, Non-clairvoyant scheduling of multiple bag-of-tasks applications, in: Proceedings of the 16th International Euro-Par Conference (Euro-Par'10), Vol. 6271 of LNCS, Springer, 2010, pp. 168–179.
[27] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, F. Vivien, Scheduling concurrent bag-of-tasks applications on heterogeneous platforms, IEEE Transactions on Computers 59 (2) (2010) 202–217.
[28] O. Beaumont, A. Legrand, L. Marchal, 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), IEEE Computer Society, 2004.
[29] B. Hong, V. K. Prasanna, Adaptive allocation of independent tasks to maximize throughput, IEEE Transactions on Parallel and Distributed Systems 18 (10) (2007) 1420–1435.
[30] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, Y. Robert, Centralized versus distributed schedulers multiple bag-of-tasks applications, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 698–709.
[31] S. Low, A duality model of TCP and queue management algorithms, IEEE/ACM Transactions on Networking 11 (4) (2003) 525–536.
[32] F. Kelly, A. Maulloo, D. Tan, Rate control in communication networks: shadow prices, proportional fairness and stability, Journal of the Operational Research Society 49 (1998) 237–252.
[33] D. P. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, 1992.
[34] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: Fair allocation of multiple resource types, in: Proceedings of the 8th USENIX conference on Networked systems design and implementation (NSDI'11), USENIX Association, 2011, p. 24.

[35] D. P. Bertsekas, J. N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, 1989.
[36] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, D. Towsley, Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet, IEEE/ACM Transaction on Networking 14 (6) (2006) 1260–1271.
[37] X. Lin, N. B. Shroff, Utility maximization for communication networks with multipath routing, IEEE Transactions on Automatic Control 51 (5) (2006) 766–781.
[38] IETF, Multipath TCP working group, http://datatracker.ietf.org/wg/mptcp/charter/ (05 2013).
[39] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.-Y. Le Boudec, MPTCP is not pareto-optimal: performance issues and a possible solution, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12), ACM, 2012, pp. 1–12.
[40] E. Caron, F. Desprez, DIET: A scalable toolbox to build network enabled servers on the grid, International Journal of High Performance Computing Applications 20 (3) (2006) 335–352.
[41] E. Santos-Neto, W. Cirne, F. Brasileiro, A. Lima, Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids, in: Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2004), no. 3277 in LNCS, Springer, 2004, pp. 210–232.
[42] K. Kaya, C. Aykanat, Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments, IEEE Transactions on Parallel Distributed Systems 17 (8) (2006) 883–896.
[43] R. Bertin, A. Legrand, 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), IEEE, 2008, pp. 118–125.
[44] G. Tel, Introduction to Distributed Algorithms, 2nd Edition, Cambridge University Press, New York, NY, USA, 2001.
[45] A. Legrand, M. Quinson, K. Fujiwara, 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), IEEE, 2006, pp. 385–386.
[46] C. Touati, E. Altman, J. Galtier, Generalized Nash bargaining solution for bandwidth allocation, Computer Networks 50 (17) (2006) 3242–3263.
[47] S. J. Benson, Y. Ye, Algorithm 875: DSDP5-software for semidefinite programming, ACM Transactions on Mathematical Software 34 (3) (2008) 16:1–16:20.
[48] D. C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, 2005.
[49] T. F. Jaeger, Categorical data analysis: Away from ANOVAs (transformation or not) and towards Logit Mixed Models, Journal of Memory and Language 59 (4) (2008) 434–446.
[50] R. Bertin, S. Hunold, A. Legrand, C. Touati, From flow control in multi-path networks to multiple bag-of-tasks application scheduling on grids, Tech. Rep. 7745, INRIA, companion webpage: http://mescal.imag.fr/membres/arnaud.legrand/distla_2012/. (2011).
[51] P. Coucheney, C. Touati, B. Gaujal, Fair and efficient user-network association algorithm for multi-technology wireless networks, in: Proceedings of the 28th Conference on Computer Communications (INFOCOM'2009), IEEE, 2009, pp. 2811–2815.