

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

Rémi Bertin, Arnaud Legrand, Corinne Touati

Laboratoire LIG, CNRS-INRIA Grenoble, France

May 22, 2008

- 1 Framework
- 2 Lagrangian Optimization
- 3 Simulations: Early Results

Large-scale distributed computing platforms result from the collaboration of **many users**:

- ▶ **Sharing** resources amongst users should somehow be **fair**.

Large-scale distributed computing platforms result from the collaboration of **many users**:

- ▶ **Sharing** resources amongst users should somehow be **fair**.
- ▶ The size of these systems prevents the use of centralized approaches \leadsto need for **distributed** scheduling.

Large-scale distributed computing platforms result from the collaboration of **many users**:

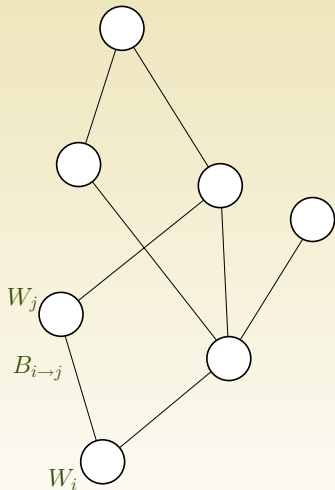
- ▶ **Sharing** resources amongst users should somehow be **fair**.
- ▶ The size of these systems prevents the use of centralized approaches \rightsquigarrow need for **distributed** scheduling.
- ▶ Task **regularity** (SETI@home, BOINC, ...) \rightsquigarrow **steady-state** scheduling.

Large-scale distributed computing platforms result from the collaboration of **many users**:

- ▶ **Sharing** resources amongst users should somehow be **fair**.
- ▶ The size of these systems prevents the use of centralized approaches \rightsquigarrow need for **distributed** scheduling.
- ▶ Task **regularity** (SETI@home, BOINC, ...) \rightsquigarrow **steady-state** scheduling.

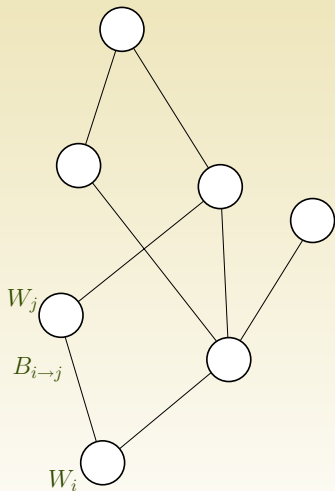
Designing a **Fair and Distributed** scheduling algorithm for this framework.

- 1 Framework
- 2 Lagrangian Optimization
- 3 Simulations: Early Results

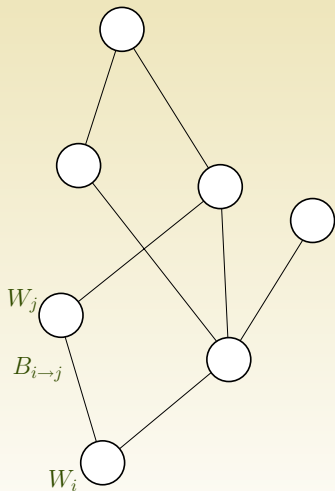


- ▶ General platform graph $G = (N, E, W, B)$.

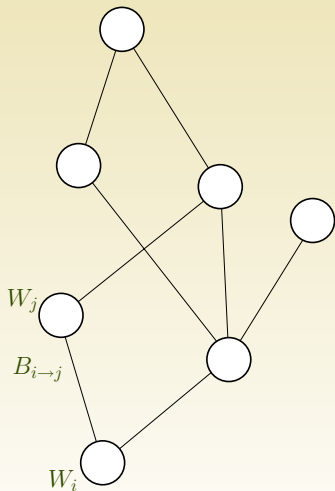
Platform Model



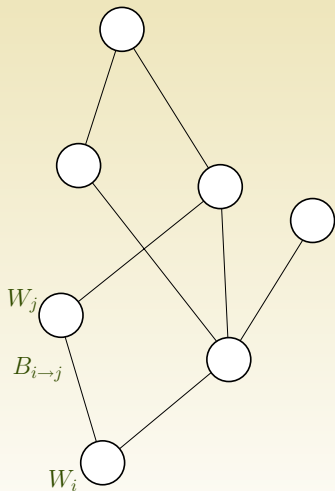
- ▶ General platform graph $G = (N, E, W, B)$.
- ▶ Speed of $P_n \in N$: W_n (in MFlops/s).
- ▶ Bandwidth of $(P_i \rightarrow P_j)$: $B_{i,j}$ (in MB/s).



- ▶ General platform graph $G = (N, E, W, B)$.
- ▶ Speed of $P_n \in N$: W_n (in MFlops/s).
- ▶ Bandwidth of $(P_i \rightarrow P_j)$: $B_{i,j}$ (in MB/s).
- ▶ Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j .



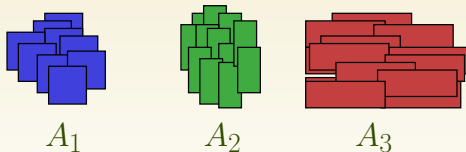
- ▶ General platform graph $G = (N, E, W, B)$.
- ▶ Speed of $P_n \in N$: W_n (in MFlops/s).
- ▶ Bandwidth of $(P_i \rightarrow P_j)$: $B_{i,j}$ (in MB/s).
- ▶ Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j .
- ▶ Communications and computations can be overlapped.



- ▶ General platform graph $G = (N, E, W, B)$.
- ▶ Speed of $P_n \in N$: W_n (in MFlops/s).
- ▶ Bandwidth of $(P_i \rightarrow P_j)$: $B_{i,j}$ (in MB/s).
- ▶ Linear-cost communication and computation model: $X/B_{i,j}$ time units to send a message of size X from P_i to P_j .
- ▶ Communications and computations can be overlapped.
- ▶ Multi-port communication model.

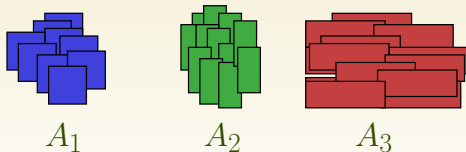
Multiple applications:

- ▶ A set A of K applications A_1, \dots, A_K .



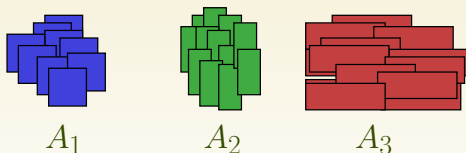
Multiple applications:

- ▶ A set A of K applications A_1, \dots, A_K .
- ▶ Each consisting in a **large number** of **same-size independent** tasks \rightsquigarrow each application is defined by a communication cost w_k (in MFlops) and a communication cost b_k (in MB).

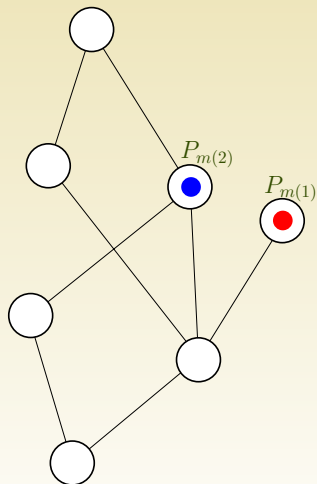


Multiple applications:

- ▶ A set A of K applications A_1, \dots, A_K .
- ▶ Each consisting in a **large number of same-size independent** tasks \rightsquigarrow each application is defined by a communication cost w_k (in MFlops) and a communication cost b_k (in MB).
- ▶ Different communication and computation demands for different applications.

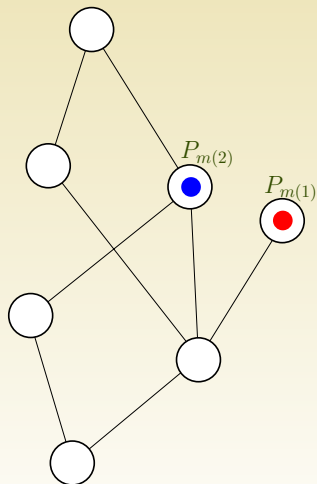


Hierarchical Deployment



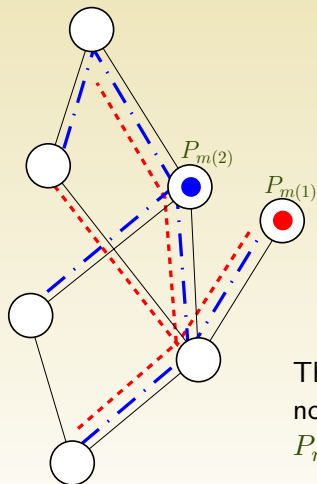
- ▶ Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k .

Hierarchical Deployment



- ▶ Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k .
- ▶ Communication are only required outwards from the master nodes: the amount of data returned by the worker is negligible.

Hierarchical Deployment



- ▶ Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k .
- ▶ Communication are only required outwards from the master nodes: the amount of data returned by the worker is negligible.
- ▶ Each application A_k is deployed on the platform as a tree.

Therefore if an application k wants to use a node P_n , all its data will use a single path from $P_{m(k)}$ to P_n denoted by $(P_{m(k)} \rightsquigarrow P_n)$.

- ▶ All tasks of a given application are **identical** and **independent**
 \rightsquigarrow we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).

Steady-State Scheduling and Utility

- ▶ All tasks of a given application are **identical** and **independent**
 \rightsquigarrow we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).
- ▶ We only need to focus on **average values in steady-state**.

Steady-State Scheduling and Utility

- ▶ All tasks of a given application are **identical** and **independent**
 \rightsquigarrow we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).
- ▶ We only need to focus on **average values in steady-state**.
- ▶ Steady-state values:

Steady-State Scheduling and Utility

- ▶ All tasks of a given application are **identical** and **independent**
 \rightsquigarrow we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).
- ▶ We only need to focus on **average values in steady-state**.
- ▶ Steady-state values:
 - ▶ Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$.

Steady-State Scheduling and Utility

- ▶ All tasks of a given application are **identical** and **independent**
 \rightsquigarrow we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).
- ▶ We only need to focus on **average values in steady-state**.
- ▶ Steady-state values:
 - ▶ Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$.
 - ▶ **Throughput** of application k : $\rho_k = \sum_{n \in N} \rho_{n,k}$.

- ▶ All tasks of a given application are **identical** and **independent**
↪ we do not really need to care about *where* and *when* (as opposed to classical scheduling problems).
- ▶ We only need to focus on **average values in steady-state**.
- ▶ Steady-state values:
 - ▶ Variables: average number of tasks of type k processed by processor n per time unit: $\rho_{n,k}$.
 - ▶ **Throughput** of application k : $\rho_k = \sum_{n \in N} \rho_{n,k}$.

Theorem 1.

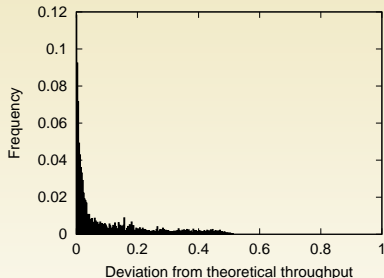
From “feasible” $\rho_{n,k}$, it is possible to build an optimal **periodic** infinite schedule (i.r. whose steady-state rates are exactly the $\rho_{n,k}$). Such a schedule is **asymptotically optimal** for the makespan.

Decentralized Scheduling

The rates $\rho_{n,k}$ are sufficient to help **simple demand-driven** scheduling algorithms.

- ▶ Dispatch incoming tasks of type k to the queues (n, k) with “proportion” $\rho_{n,k}$.
- ▶ Request tasks from your father when incoming queue sizes get below a fixed threshold.

$$\text{Deviation} = \frac{\rho_k^{(th)} - \rho_k^{(exp)}}{\rho_k^{(th)}}$$



↪ We can focus on finding the $\rho_{n,k}$.

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.
- ▶ It has been shown that different values of U_k leads to different kind of fairness. Typically, $U_k(\rho_k) = \log(\rho_k)$ (proportional fairness) or $U_k(\rho_k) = \rho_k^\alpha / (1 - \alpha)$ (α -fairness).

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.
- ▶ It has been shown that different values of U_k leads to different kind of fairness. Typically, $U_k(\rho_k) = \log(\rho_k)$ (proportional fairness) or $U_k(\rho_k) = \rho_k^\alpha / (1 - \alpha)$ (α -fairness).

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.
- ▶ It has been shown that different values of U_k leads to different kind of fairness. Typically, $U_k(\rho_k) = \log(\rho_k)$ (proportional fairness) or $U_k(\rho_k) = \rho_k^\alpha / (1 - \alpha)$ (α -fairness).
- ▶ Maximize $\sum_k \log(\rho_k)$ under the constraints:

$$\begin{aligned}\rho_k &= \sum_n \rho_{n,k} \\ \forall n, \quad \sum_k \rho_{n,k} w_k &\leq W_n \\ \forall (P_i \rightarrow P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \rightarrow P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k &\leq B_{i,j}\end{aligned}$$

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.
- ▶ It has been shown that different values of U_k leads to different kind of fairness. Typically, $U_k(\rho_k) = \log(\rho_k)$ (proportional fairness) or $U_k(\rho_k) = \rho_k^\alpha / (1 - \alpha)$ (α -fairness).
- ▶ Maximize $\sum_k \log(\rho_k)$ under the constraints:

$$\begin{aligned} \rho_k &= \sum_n \rho_{n,k} \\ \forall n, \quad \sum_k \rho_{n,k} w_k &\leq W_n \\ \forall (P_i \rightarrow P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \rightarrow P_j) \in (P_m(k) \rightsquigarrow P_n)}} \rho_{n,k} b_k &\leq B_{i,j} \end{aligned}$$

- ▶ Can be solved in polynomial time with **semi-definite programming** [Touati.et.al.06]. It is very centralized though.

Can we solve it in a distributed way?

Utility and Optimization Problem

- ▶ Let $U_k(\rho_k)$ be the utility associated to application k . We aim at maximizing $\sum_{k \in K} U_k(\rho_k)$.
- ▶ It has been shown that different values of U_k leads to different kind of fairness. Typically, $U_k(\rho_k) = \log(\rho_k)$ (**proportional fairness**) or $U_k(\rho_k) = \rho_k^\alpha / (1 - \alpha)$ (α -fairness).
- ▶ Maximize $\sum_k \log(\rho_k)$ under the constraints:

$$\begin{aligned}\rho_k &= \sum_n \rho_{n,k} \\ \forall n, \quad \sum_k \rho_{n,k} w_k &\leq W_n \\ \forall (P_i \rightarrow P_j), \quad \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \rightarrow P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \rho_{n,k} b_k &\leq B_{i,j}\end{aligned}$$

- ▶ Can be solved in polynomial time with **semi-definite programming** [Touati.et.al.06]. It is very centralized though.

Can we solve it in a **distributed** way?

- 1 Framework
- 2 Lagrangian Optimization
- 3 Simulations: Early Results

- ▶ Designed to solve **non linear optimization** problems:
 - ▶ Let $\alpha \rightarrow f(\alpha)$ be a function to maximize.
 - ▶ Let $(C_i(\alpha) \geq 0)_{i \in [1..n]}$ be a set of n constraints.
 - ▶ We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geq 0, \text{ and } \alpha \geq 0 \end{cases}$$

- ▶ Designed to solve **non linear optimization** problems:
 - ▶ Let $\alpha \rightarrow f(\alpha)$ be a function to maximize.
 - ▶ Let $(C_i(\alpha) \geq 0)_{i \in [1..n]}$ be a set of n constraints.
 - ▶ We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geq 0, \text{ and } \alpha \geq 0 \end{cases}$$

- ▶ The **Lagrangian function**: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$.

- ▶ Designed to solve **non linear optimization** problems:
 - ▶ Let $\alpha \rightarrow f(\alpha)$ be a function to maximize.
 - ▶ Let $(C_i(\alpha) \geq 0)_{i \in [1..n]}$ be a set of n constraints.
 - ▶ We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geq 0, \text{ and } \alpha \geq 0 \end{cases}$$

- ▶ The **Lagrangian function**: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$.
- ▶ The **dual functional**: $d(\lambda) = \max_{\alpha \geq 0} \mathcal{L}(\alpha, \lambda)$.

- ▶ Designed to solve **non linear optimization** problems:
 - ▶ Let $\alpha \rightarrow f(\alpha)$ be a function to maximize.
 - ▶ Let $(C_i(\alpha) \geq 0)_{i \in [1..n]}$ be a set of n constraints.
 - ▶ We wish to solve:

$$(P) \begin{cases} \text{maximize } f(\alpha) \\ \forall i \in [1..n], C_i(\alpha) \geq 0, \text{ and } \alpha \geq 0 \end{cases}$$

- ▶ The **Lagrangian function**: $\mathcal{L}(\alpha, \lambda) = f(\alpha) - \sum_{i \in [1..n]} \lambda_i C_i(\alpha)$.
- ▶ The **dual functional**: $d(\lambda) = \max_{\alpha \geq 0} \mathcal{L}(\alpha, \lambda)$.
- ▶ Under some weak hypothesis, solving (P) is equivalent to solve the **dual problem**:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geq 0 \end{cases}$$

- ▶ Designed to solve **non linear optimization** problems:

▶ Let $\alpha \mapsto f(\alpha)$ be a function to maximize

So what?..

- ▶ Two **coupled** problems with **simple constraints**.
- ▶ The structure of constraints is transposed to (D) and a **gradient descent** algorithm is a natural way to solve these two problems.
- ▶ This technique has been used successfully for network resource sharing [Kelly.98], TCP analysis [Low.03], flow control in multi-path network [Hang.et.al.03], ...

the **dual problem**:

$$(D) \begin{cases} \text{minimize } d(\lambda) \\ \lambda \geq 0 \end{cases}$$

- ▶ What does the Lagrangian function look like ?

$$\mathcal{L}(\alpha, \lambda, \mu) = \sum_{k \in K} \log \left(\sum_i \varrho_{i,k} \right) + \sum_i \lambda_i \left(W_i - \sum_k \varrho_{i,k} w_k \right) \\ + \sum_{(P_i \rightarrow P_j)} \mu_{i,j} \left(B_{i,j} - \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \rightarrow P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \right)$$

Trying to use Lagrangian optimization

- ▶ What does the Lagrangian function look like ?

$$\mathcal{L}(\alpha, \lambda, \mu) = \sum_{k \in K} \log \left(\sum_i \varrho_{i,k} \right) + \sum_i \lambda_i \left(W_i - \sum_k \varrho_{i,k} w_k \right) + \sum_{(P_i \rightarrow P_j)} \mu_{i,j} \left(B_{i,j} - \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \rightarrow P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \right)$$

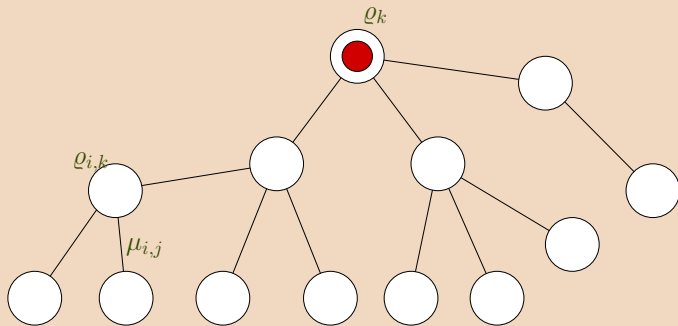
- ▶ Remember, we want to compute $\min_{\lambda, \mu \geq 0} \max_{\varrho \geq 0} \mathcal{L}(\alpha, \lambda, \mu)$. We can solve this problem by simply doing a “alternate” gradient descent (I’m skipping a few details here to keep it simple and just present the general idea):

$$\begin{cases} \varrho_{i,k} & \leftarrow \varrho_{i,k} + \gamma \frac{\partial \mathcal{L}}{\partial \varrho_{i,k}} \\ \lambda_i & \leftarrow \lambda_i - \gamma \frac{\partial \mathcal{L}}{\partial \lambda_i} \\ \mu_{i,j} & \leftarrow \mu_{i,j} - \gamma \frac{\partial \mathcal{L}}{\partial \mu_{i,j}} \end{cases}$$

- ▶ $q_{i,k}$ is “private” to the agent of application k running on node i .
- ▶ λ_i is attached to node i and $\mu_{i,j}$ is attached to $(P_i \rightarrow P_j)$. λ_i and $\mu_{i,j}$ are called shadow variables or **shadow prices**. They can naturally thought of as the *price to pay to use the corresponding resource*.
- ▶ A **gradient descent** algorithm on the primal-dual problem can thus be seen as a **bargain** between applications and resources.
- ▶ We need to find an efficient way to implement this bargain, i.e., to compute the update. To this end, the following quantities are useful and easy to compute via recursive propagation:

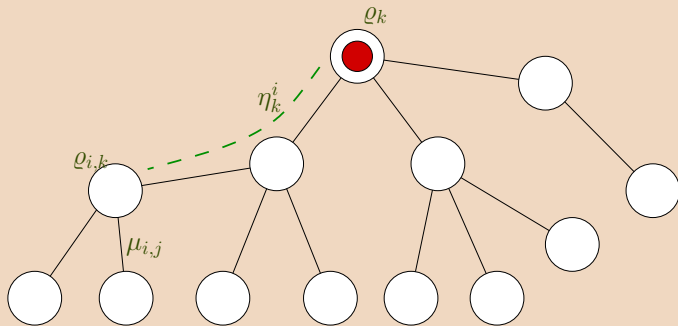
$$\left\{ \begin{array}{l} \sigma_k^n = \sum_{p \text{ such that } n \in (P_m(k) \rightsquigarrow P_p)} q_{p,k} \\ \eta_k^n = \sum_{(P_i \rightarrow P_j) \in (P_m(k) \rightsquigarrow P_n)} \mu_{i,j} \end{array} \right. \begin{array}{l} \left\{ \begin{array}{l} \text{aggregate throughput} \\ \text{of a subtree.} \end{array} \right. \\ \left\{ \begin{array}{l} \text{aggregate communication} \\ \text{price} \end{array} \right. \end{array}$$

Hierarchical deployment



$$\left\{ \begin{array}{l} \sigma_k^n = \sum_{p \text{ such that } n \in (P_{m(k)} \rightsquigarrow P_p)} \varrho_{p,k} \\ \eta_k^n = \sum_{(P_i \rightarrow P_j) \in (P_{m(k)} \rightsquigarrow P_n)} \mu_{i,j} \end{array} \right. \begin{array}{l} \left\{ \begin{array}{l} \text{aggregate throughput} \\ \text{of a subtree.} \end{array} \right. \\ \left\{ \begin{array}{l} \text{aggregate communication} \\ \text{price} \end{array} \right. \end{array}$$

Hierarchical deployment



$$\left\{ \begin{array}{l} \sigma_k^n = \sum_{p \text{ such that } n \in (P_m^{(k)} \rightsquigarrow P_p)} \varrho_{p,k} \\ \eta_k^n = \sum_{(P_i \rightarrow P_j) \in (P_m^{(k)} \rightsquigarrow P_n)} \mu_{i,j} \end{array} \right. \begin{array}{l} \left\{ \begin{array}{l} \text{aggregate throughput} \\ \text{of a subtree.} \end{array} \right. \\ \left\{ \begin{array}{l} \text{aggregate communication} \\ \text{price} \end{array} \right. \end{array}$$

Prices and rates can thus be propagated and aggregated to perform the following updates:

$$p_k^i(t+1) \leftarrow b_k \eta_k^i(t) + w_k \lambda_i(t)$$

$$\rho_k(t+1) \leftarrow \sigma_k^{m(k)}(t+1)$$

$$\rho_{i,k}(t+1) \leftarrow [\rho_{i,k}(t) + \gamma_\rho (U'_k(\rho_k(t)) - p_k^i(t))]^+$$

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

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

- ▶ This algorithm is *fully distributed* and converges to the *optimal* solution **provided a good choice** of γ_ρ , γ_λ and γ_μ is done.
- ▶ This algorithm *seamlessly adapts* to application/node arrival and to load variations.

- 1 Framework
- 2 Lagrangian Optimization
- 3 Simulations: Early Results**

- ▶ The simulator is SimGrid.

Experimental Setting

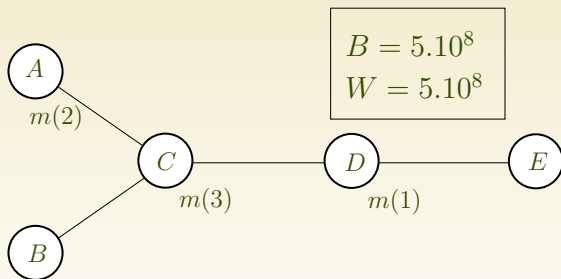
- ▶ The simulator is SimGrid.
- ▶ Fully synchronous gradient.

Experimental Setting

- ▶ The simulator is SimGrid.
- ▶ Fully synchronous gradient.
- ▶ Checking the correctness of the results using semi-definite programming.

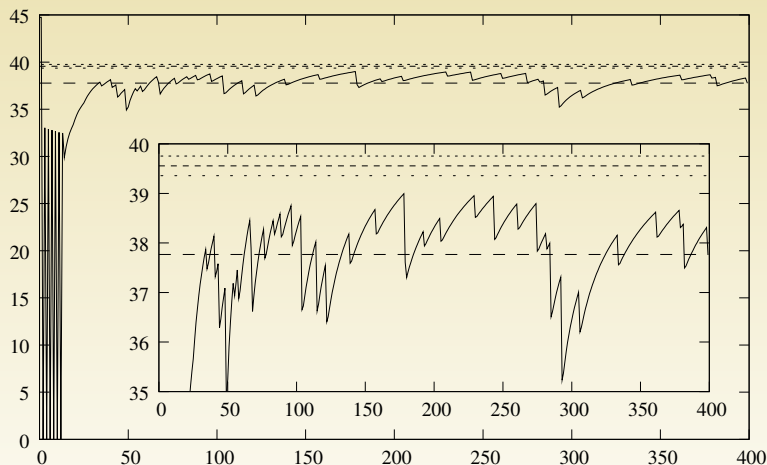
Experimental Setting

- ▶ The simulator is SimGrid.
- ▶ Fully synchronous gradient.
- ▶ Checking the correctness of the results using semi-definite programming.
- ▶ Very simple platform and applications:



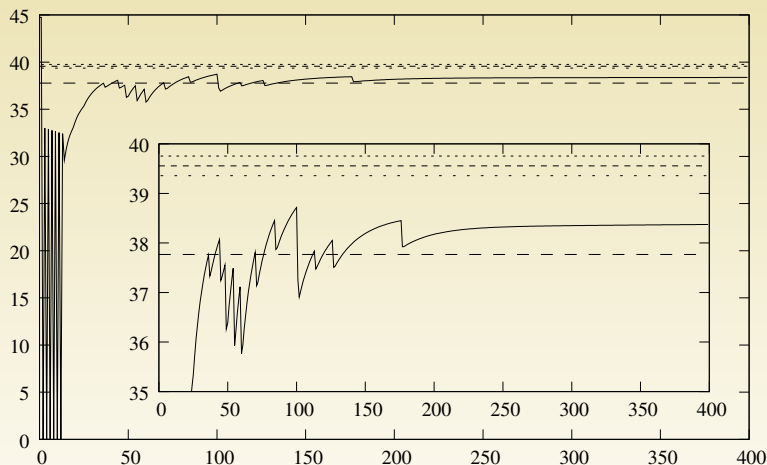
We used three kinds of applications of respective (b, w) : $(1000, 5000)$, $(2000, 800)$, and $(1500, 1500)$.

Basic Version of the Algorithm



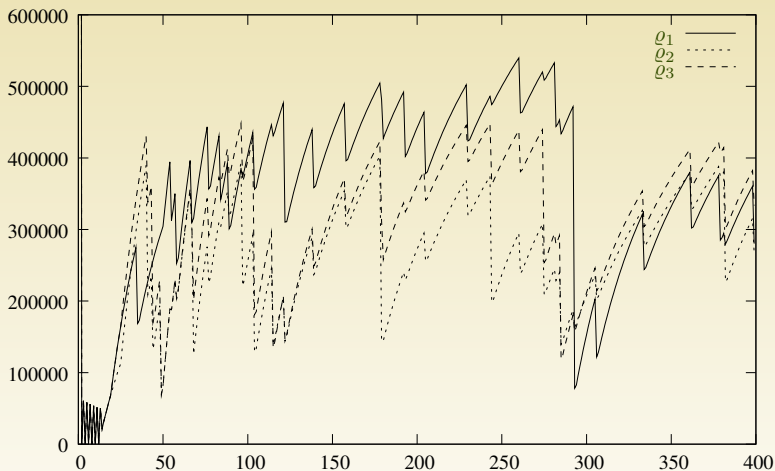
Objective function $\sum_k \log \rho_k$: numerical instabilities and global inefficiencies.

Basic Version of the Algorithm



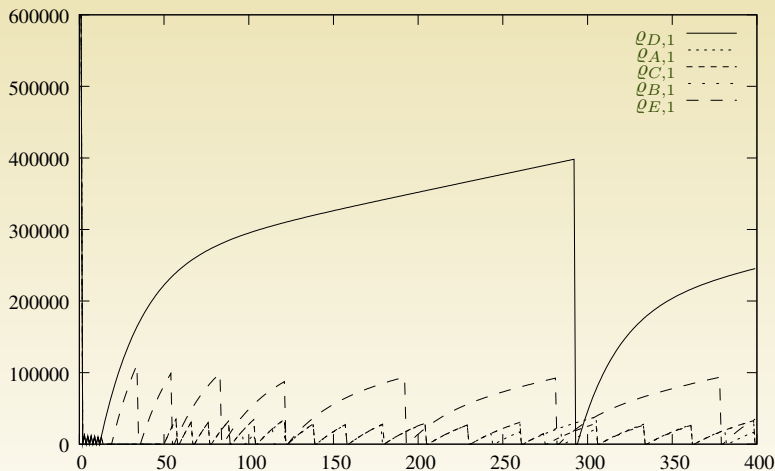
Objective function $\sum_k \log \rho_k$: using a smaller steps $\gamma_\rho \rightsquigarrow$ no more instability but slow convergence.

Basic Version of the Algorithm



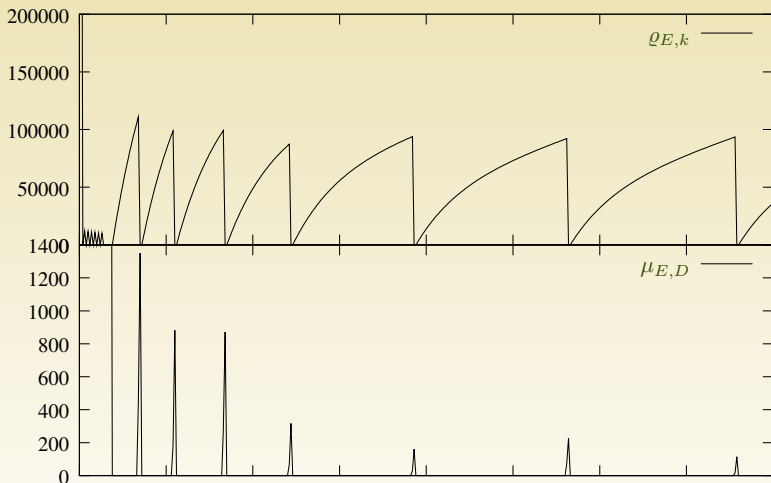
Throughput of each of the three applications: between two iterations, a decrease or increase of magnitude five or more can happen!

Basic Version of the Algorithm



Detailing the rates for application 1.

Basic Version of the Algorithm



Correlation between the rate of an application on a given node and the price it experiences.

Scaling!

The original update equation for ϱ is:

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

A small value of ϱ leads to huge updates and thus to severe oscillations.

The original update equation for ϱ is:

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

A small value of ϱ leads to huge updates and thus to severe oscillations. This is a known issue and, as mentioned in [Hang.et.al.03], one can normalize as follows:

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

Unfortunately, it does not help (the previous experiments actually use this normalized update). It merely avoids division by 0 but is insufficient to damp oscillations.

The original update equation for ϱ is:

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

A small value of ϱ leads to huge updates and thus to severe oscillations. This is a known issue and, as mentioned in [Hang.et.al.03], one can normalize as follows:

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

Unfortunately, it does not help (the previous experiments actually use this normalized update). It merely avoids division by 0 but is insufficient to damp oscillations.

Updating ϱ has an impact on the prices λ and μ , which in turn impact on the ϱ 's update.

The original update equation for ϱ is:

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

A small value of ϱ leads to huge updates and thus to severe oscillations. This is a known issue and, as mentioned in [Hang.et.al.03], one can normalize as follows:

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

Unfortunately, it does not help (the previous experiments actually use this normalized update). It merely avoids division by 0 but is insufficient to damp oscillations.

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.

Scaling Again!

Assume that we have reached the equilibrium. Then increase λ_i by $\Delta\lambda_i$. Then:

$$\Delta\rho_{i,k} = -\gamma_{\rho}^{(2)} w_k \Delta\lambda_i \rho_k.$$

Scaling Again!

Assume that we have reached the equilibrium. Then increase λ_i by $\Delta\lambda_i$. Then:

$$\Delta\rho_{i,k} = -\gamma_\rho^{(2)} w_k \Delta\lambda_i \rho_k.$$

In turn, such a variation incurs a variation of λ_i :

$$\sum_k \gamma_\lambda \cdot w_k \cdot \Delta\rho_{i,k} = \Delta\lambda_i \cdot \left(\sum_k \gamma_\lambda \cdot \gamma_\rho^{(2)} w_k^2 \rho_k \right).$$

Scaling Again!

Assume that we have reached the equilibrium. Then increase λ_i by $\Delta\lambda_i$. Then:

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

In turn, such a variation incurs a variation of λ_i :

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

Thus, the solution of our gradient is stable only if

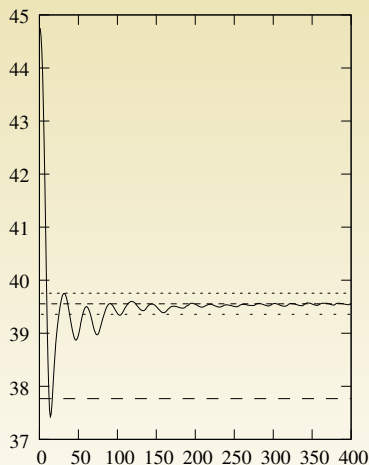
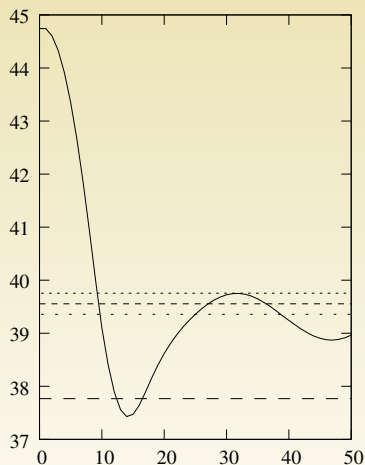
$$\sum_k \gamma_{\lambda} \cdot \gamma_{\varrho}^{(2)} w_k^2 \varrho_k < 1.$$

Therefore, λ 's update should be replaced by

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

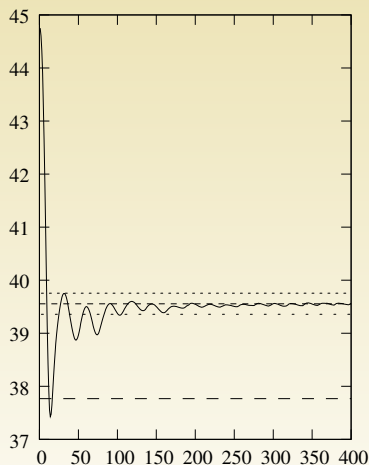
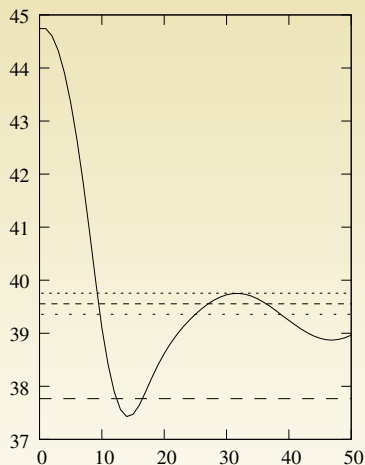
It doesn't hurt and similar scaling can be done for the μ 's.

Scaled Version of the Algorithm



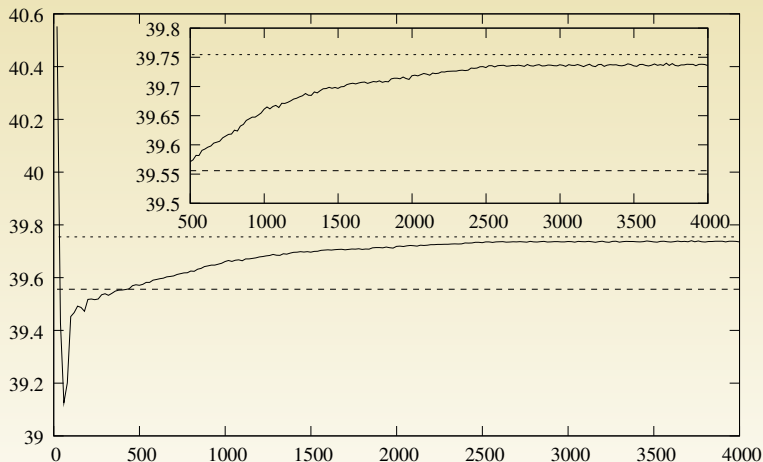
The oscillations, due to a really badly chosen initialization value quickly vanish (left graph).

Scaled Version of the Algorithm



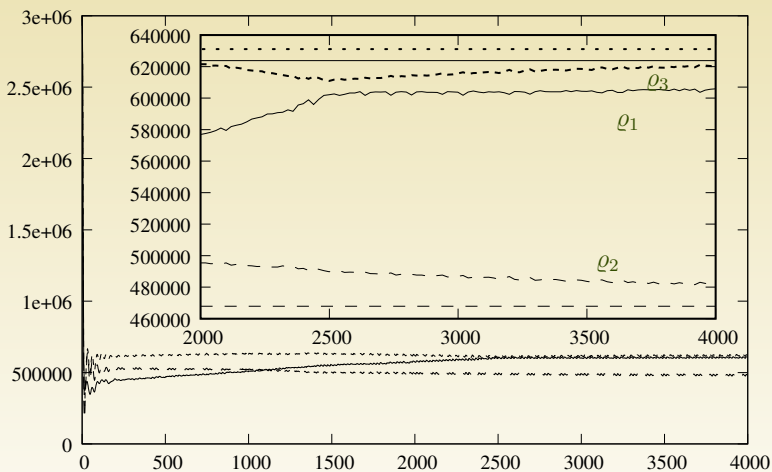
The algorithm almost instantly reaches a decent value (5% of the optimal value after 17 iterations), and relatively quickly to a good value (1% of the optimal value after 83 iterations) (right plot).

Scaled Version of the Algorithm



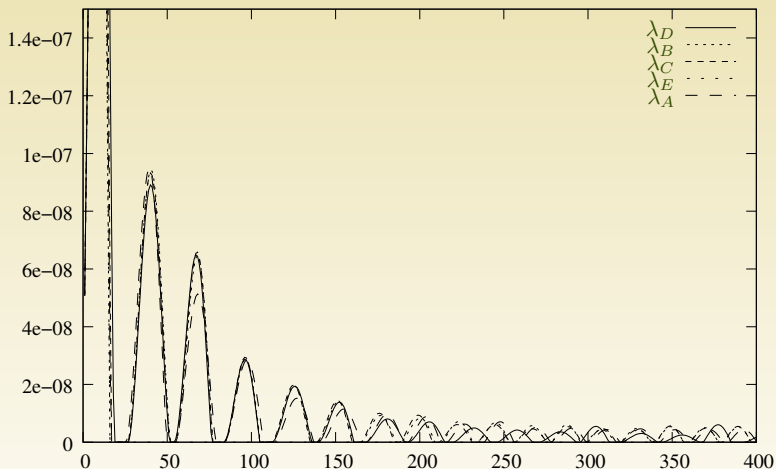
High number of iterations: after 498 iterations, the performance remains higher than 99.5% of the optimal and still further increase with the number of iterations.

Scaled Version of the Algorithm



Convergence of ρ_i with $i = 1..3$: no more oscillations occur. The throughput of each application slowly converges to their “optimal” values.





Scaled Version of the Algorithm



Prices evolve smoothly. As the number of iterations increase, they converge to their optimal value while remaining positive, meaning that the resources they refer to is neither under utilized nor overloaded.

- ▶ Not enough time to present related work but this approach is very inspired by Low's work [Hang.et.al.03] on flow control in multi-path network.
- ▶ The setting (BoT applications, grids) is different though and new problem arise.
- ▶ The resulting algorithms are different (few sources and many sinks here).
- ▶ The convergence issue is mainly due to the fact that the resource usage is not homogeneous (each application has its own w_k and b_k). The previous scaling is effective and easy to implement.

- ▶ There may be situations where the previous scaling may not be sufficient though. When the optimal throughput of the applications do not have the same order of magnitude, it may be necessary for each application to have its own step size $\gamma_{\varrho}^{(2)}$. We may need to find auto-scaling for the ϱ 's update as well.
- ▶ Wouldn't this "hand-made" scaling correspond to another well-known scaling? Then, maybe there would be better scaling scheme. . .
- ▶ The present convergence study is rather limited in term of scalability. . .
- ▶ We target grid or desktop-grid-like platforms. What if the number of application has the same order of magnitude as the number of participants in the system (like in a peer-to-peer system)? Would the steady-state approach still make sense (response-time based metrics like stretch. . .)?
- ▶ We rely on steady-state. How does such a system react to high churn?

-  Frank Kelly, Aman Maulloo, and David Tan.
Rate control in communication networks: shadow prices, proportional fairness and stability.
Journal of the Operational Research Society, 49:237–252, 1998.
-  Steven Low.
A duality model of TCP and queue management algorithms.
IEEE/ACM Transactions on Networking, 11(4):525–536, 2003.
-  Corinne Touati, Eitan Altman, and Jérôme Galtier.
Generalized Nash bargaining solution for bandwidth allocation.
Computer Networks, 50(17):3242–3263, December 2006.
-  Wei-Hua Wang, Marimuthu Palaniswami, and Steven Low.
Optimal flow control and routing in multi-path networks.
Performance Evaluation, 52:119–132, 2003.