Available online at www.sciencedirect.com
ScienceDirect
Sustainable Computing 00 (2014) 1–18
Suscom Logo

Power-aware replica placement in tree networks with multiple servers per client

Guillaume Aupy[a,*], Anne Benoit[a], Matthieu Journault[b], Yves Robert[a,c]

[a] *LIP, École Normale Supérieure de Lyon, CNRS, Université de Lyon & INRIA, France*
[b] *École Normale Supérieure de Cachan, France*
[c] *University of Tennessee Knoxville, USA*

Abstract

In this paper, we revisit the well-studied problem of replica placement in tree networks. Rather than minimizing the number of servers needed to serve all client requests, we aim at minimizing the total power consumed by these servers. In addition, we use the most general (and powerful) server assignment policy, where the requests of a client can be served by multiple servers located in the (unique) path from this client to the root of the tree. We consider multi-modal servers that can operate at a set of discrete speeds, using the dynamic voltage and frequency scaling (DVFS) technique. The optimization problem is to determine an optimal location of the servers in the tree, as well as the speed at which each server is operated. A major result is the NP-completeness of this problem, to be contrasted with the minimization of the number of servers, which has polynomial complexity. Another important contribution is the formulation of a Mixed Integer Linear Program (MILP) for the problem, together with the design of several polynomial-time heuristics. We assess the efficiency of these heuristics by simulation. For mid-size instances (up to 30 nodes in the tree), we evaluate their absolute performance by comparison with the optimal solution (obtained via the MILP). The most efficient heuristics provide satisfactory results, within 20% of the optimal solution.

Keywords: replica placement; tree network; multiple servers; power-aware algorithm; complexity; discrete speeds.

*Corresponding author. Address: Laboratoire LIP, ENS Lyon, 69364 Lyon Cedex 07. Phone: +33472728352. Fax:+33472728080. Email: Guillaume.Aupy@ens-lyon.fr

1. Introduction

In this paper, we revisit the well-studied problem of replica placement in tree networks. Replica placement in tree networks is an important problem [1, 2, 3], with a broad spectrum of applications, such as electronic, ISP, or VOD service delivery (see [4, 1, 5] and additional references in [2]). The problem is the following: one is given a tree-shaped network where clients are periodically issuing requests to be satisfied by servers. The clients are known (both their position in the tree and their number of requests per time unit), while the number and location of the replicas (also called servers) are to be determined. Clients are leaves of the tree, and requests can be served by one or several internal nodes. Note that the distribution tree (clients and nodes) is fixed in the approach.

Initially, there is no replica; when a node is equipped with a replica, it can process a number of requests, up to its capacity limit. Nodes equipped with a replica, also called servers, can only serve clients located in their subtree (so that the root, if equipped with a replica, can serve any client); this restriction is usually adopted to enforce the hierarchical nature of the target application platforms, where a node has knowledge only of its parent and children in the tree. More precisely, there are three classical policies to serve the requests of a client [3]: (i) *Closest*: All requests of a client must be served by the first server located in the path from this client to the root; (ii) *Single*: All requests of a client must be served by a single server, located anywhere in the path from this client to the root; and (iii) *Multiple*: The requests of a client can be served by several servers, all located in the path from this client to the root. For instance in the *Multiple* policy, half the requests of a client can be served by one server, and the other half by another server located higher in this path. In this paper, we study the *Multiple* policy, because it is the most flexible, hence it will lead to the most efficient solution in terms of both the number of servers and total consumed power. The *Multiple* strategy can safely be used in all applications, because in the classical replica placement problem, it is assumed that (i) the servers are identical and capable of serving any client; and (ii) all requests have same size and cost.

The usual optimization objective in the literature is the number of servers needed to serve all requests. However, minimizing the total power consumed by the servers has recently become a very important objective, both for economic and environmental reasons [6]. To help reduce power dissipation, multi-modal servers are used: each server has a *discrete* number of predefined speeds, which correspond to different voltages that the server can be subjected to. State-of-the-art processors can only be operated with a restricted number of voltage levels, hence with a few speeds [7, 8]. The power consumption is the sum of a static part (the cost for a server to be on and operated) and a dynamic part. This dynamic part is a strictly convex function of the server speed, so that the execution of a given amount of work costs more power if a server runs at a higher speed [8]. More precisely, a server operated at speed s dissipates s^3 watts [9, 10, 11]. Faster speeds allow servers to handle more requests per time unit, but at the price of a much higher (supra-linear) power consumption.

A major contribution of this paper is to show that minimizing power consumption is an NP-complete problem, even if the servers are already placed in the network (and without static power). This is to be contrasted with the polynomial complexity of minimizing the number or servers [3]. Another major contribution is the design of a set of heuristics to minimize power consumption. These heuristics work in two steps: (i) server placement and (ii) request assignment. The placement step relies on an interesting theoretical result: given a fixed set of servers that should all be used, and assuming continuous speeds, it is possible to optimally assign the requests to these servers in polynomial time. We can therefore easily derive a greedy algorithm to place the servers in the continuous case, because for a given placement, we can directly compute the corresponding optimal power consumption. Of course, assuming continuous speeds is not realistic, but it is a handy simplification of the problem: with continuous speeds, once requests are assigned to servers, each server can operate just at the right speed, namely the sum of its requests, so that selecting the server speeds is immediate. With discrete speeds, the problem is more challenging and may well lead to re-assign the requests, for a given placement of servers. To see this, we start from the solution with continuous speeds (including the greedy placement and the optimal request assignment). Let r be the number of requests processed by a given server N in the solution with continuous speeds. With discrete speeds, we have to use the smallest speed s that is larger than r, thereby losing a lot of power if the difference $s-r$ is large. If it is the case, we can try and re-assign some requests to another server N' located upper in the path from N to the tree root. There would then remain only s' requests to be served by N, where s' is the largest speed that is smaller than r: this saves power locally by avoiding the large $s-r$ gap, but we have to re-assign $r-s'$ requests

to another server, and this has a cost that should be balanced with the local gain. Such trade-off decisions are exactly those taken in the request assignment step of the heuristics.

To the best of our knowledge, this paper is the first to propose heuristics for power minimization with multiple servers, hence we cannot use any heuristics from the literature as reference. However, we have derived a Mixed Integer Linear Program (MILP) to compute the optimal solution to the power minimization problem. Using this linear program has (potentially) an exponential cost, but it enables us to assess the absolute performance of the heuristics, at least for small-size problems.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 is devoted to a precise statement of the framework. Section 4 assesses the complexity of the power minimization problem, through an intricate NP-completeness proof. This section also provides the MILP to compute the optimal solution. Section 5 introduces several heuristics to solve the problem. The placement step is an incremental greedy procedure, whose evaluation is based on the optimal solution for request assignment with fixed servers, when assuming continuous speeds. Section 6 reports simulation results and comparisons of the heuristics, together with their absolute performance evaluation: the distance to the optimal solution is computed through the linear program for instances with up to 30 servers.

2. Related work

Many papers considering the replica placement problem deal with general graphs, while we focus in this work on tree networks. In the problem with a general graph network, it is already difficult to decide which spanning tree to use, in order to optimize some global objective function. A survey of work targeting performance issues can be found in [12]. Recently, some work start to tackle energy-related problems. For instance, in [13], the authors discuss thermal and power-aware task scheduling and data placement heuristics, in the context of a Hadoop system. All problems are NP-hard, and there is no tree structure but rather a set of racks, and a set of data nodes per rack.

For tree networks, a large effort has been spent to optimize the performance of replica placements, assuming that the spanning tree was given, or that the network had a tree structure initially. Most work has focused on the *Closest* policy, where a client has to be served by the closest server on the path towards the root of the tree, see for instance [1, 5]. Kalpakis et al. [4] studied a variant with bi-directional links, and therefore the tree structure may not be respected anymore, and a client may be served by a node that is not its ancestor in the tree. While the problem with a tree structure has polynomial complexity, the bi-directional problem becomes NP-complete.

Following this line of work, we had investigated in our previous work [14] the complexity of the power-aware replica placement problem with the *Closest* policy, and proved that the problem becomes NP-complete when the objective is to minimize the total power consumption. We considered servers with several distinct possible speeds, and a server operating at a given speed consumes a power composed of a static part and a dynamic part proportional to the cube of the speed. We keep the same model in this paper, because it is a classical model extensively used when considering dynamic voltage and frequency scaling (DVFS) technique [9, 10, 11, 15].

The *Multiple* policy is more flexible than *Closest* because it loosens placement rules: the requests of a client can be processed by several servers located anywhere in the path from the client to the root. Similarly to the *Closest* policy, the problem of minimizing the cost of the replica placement can be solved in polynomial time [3]. However, we are not aware of any other work aiming at optimizing the power consumption on tree networks for this *Multiple* policy.

3. Framework

This section is devoted to a precise statement of the framework. We start with a description of the replica placement problem. Then we detail the power consumption model. Finally, we state the objective function, and the corresponding optimization problems. Notations are summarized in Table 1.

$\mathcal{C}$	Set of clients
$\mathcal{N}$	Set of nodes
$\mathcal{R}$	Set of replicas
r_i	Number of requests of client i
$r_{i,j}$	Number of requests of client i executed on replica j
w_j	Number of requests executed on replica j
K	Number of speeds
$\{s_1, \ldots, s_K\}$	Set of speeds
$\mathcal{P}_{\text{static}}$	Static power
$\mathcal{P}(j)$	Power for j^{th} replica
$\mathcal{P}(\mathcal{R})$	Total power consumption

Table 1: Table of main notations.

3.1. Replica placement

We consider a distribution tree whose nodes are partitioned into a set of clients $\mathcal{C}$, and a set of nodes, $\mathcal{N}$. The clients are leaf nodes of the tree, while $\mathcal{N}$ is the set of internal nodes. Each client $i \in \mathcal{C}$ (leaf of the tree) is sending r_i requests per time unit to a database object. Internal nodes equipped with a replica (also called *servers*) can process requests from clients in their subtree. If a server $j \in \mathcal{N}$ is operated at speed s_j, then it can process up to s_j requests per time unit. Both the r_i's and the s_j's are assumed to take rational values. Note that it would be easy to allow *client-nodes* that play both the rule of a client and of a node (possibly a server), by dividing such a node into an internal node and a leaf in the tree.

For each client $i \in \mathcal{C}$ and each node $j \in \mathcal{N}$, $r_{i,j}$ is the number of requests from client i processed by server j. We must have $\sum_{j\in\mathcal{N}} r_{i,j} = r_i$ for all $i \in \mathcal{C}$, i.e., all requests are processed. Furthermore, a server cannot process more requests than its assigned speed, i.e., $w_j = \sum_{i\in\mathcal{C}} r_{i,j} \leq s_j$ for all $j \in \mathcal{N}$, where w_j is the load of server j. The set of replicas is defined as $\mathcal{R} = \{j \in \mathcal{N} | \exists i \in \mathcal{C}\ ,\ r_{i,j} > 0\}$.

3.2. Power consumption model

We (realistically) consider discrete speeds. Servers may operate only at a set $\{s_1, \ldots, s_K\}$ of different (rational) speeds, depending upon the number of requests that they have to process per time unit. We assume that $0 \leq s_1 \leq \cdots \leq s_K$, and therefore no server can handle more than s_K requests. A server with a load w will therefore operate at speed s_k, where $s_{k-1} < w \leq s_k$ (letting $s_0 = -1$ for the limit case). The power consumption of a server $j \in \mathcal{R}$ operated at speed $s(j)$ obeys the classical model [14, 9, 10, 11, 15]:

$$\mathcal{P}(j) = \mathcal{P}_{\text{static}} + s(j)^3,$$

where $\mathcal{P}_{\text{static}}$ is the static power consumption that does not depend on the speed, and $s(j)^3$ is the dynamic part of the power consumption. The total power consumption $\mathcal{P}(\mathcal{R})$ of the solution is the sum of the power consumption of all server nodes:

$$\begin{aligned}\mathcal{P}(\mathcal{R}) &= \sum_{j\in\mathcal{R}} \mathcal{P}(j) = \sum_{j\in\mathcal{R}} (\mathcal{P}_{\text{static}} + s(j)^3) \\ &= |\mathcal{R}| \times \mathcal{P}_{\text{static}} + \sum_{j\in\mathcal{R}} s(j)^3, \qquad (1)\end{aligned}$$

where $|\mathcal{R}|$ is the total number of servers in the solution.

3.3. Optimization problems

The main optimization problem is the DISCRETE problem: given a distribution tree (with a number of requests per client), decide where to place the servers, and how to distribute client requests among them (which can also be seen as assigning the speed of each server), in order to minimize the total power consumption.

We also consider the sub-problem where the servers are already placed in the tree, DISCRETE-PLACED. The goal is then only to decide how to distribute requests among servers, hence at which speed to operate each server, in order to minimize the total power consumption.

4. Complexity results

In this section, we establish the NP-completeness of DISCRETE and DISCRETE-PLACED (Section 4.1). Then in Section 4.2, we provide a Mixed Integer Linear Program (MILP) to solve DISCRETE.

4.1. NP-completeness

Theorem 1. *The* DISCRETE *and* DISCRETE-PLACED *problems are NP-complete, even with* $\mathcal{P}_{static} = 0$.

Proof. Consider the decision problem associated to DISCRETE: given a tree network with a set of requests, a set of server speeds, and a bound K on power consumption, can we find a placement of servers and an assignment of the requests to these servers so that the bound K on power consumption is met? The problem is clearly in NP: given the servers and the request assignments, we can compute the load of each server, select the smallest possible speed and compute the total power consumption in polynomial time.

To establish the completeness, we use a reduction from 2-Partition [16]. We consider an instance $\mathcal{I}_1$ of 2-Partition: given n strictly positive integers $a_1, \ldots, a_n$, does there exist a subset I of $\{1, \ldots, n\}$ such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$? Let $S = \frac{1}{2}\sum_{i=1}^{n} a_i$. We further assume, without loss of generality, that S is an integer, and that $a_i < S$ for $1 \leq i \leq n$.

We build the following instance $\mathcal{I}_2$ of the DISCRETE problem:

- $\mathcal{P}_{\text{static}} = 0$
- We first define two constants (see below for an intuitive explanation):

$$X = 2n\left(7n^2S\right)^4, \qquad (2)$$

$$M = 7n^2X^2. \qquad (3)$$

- There are $K = 2n + 1$ possible speeds for servers:

  for $1 \leq i \leq n$,
  $$\begin{cases} s_i = iX \\ \tilde{s}_i = iX\left(1 + \frac{1}{3}\frac{Ma_i}{(iX)^3} - \frac{1}{9}\left(\frac{Ma_i}{(iX)^3}\right)^2 + \frac{5}{81}\left(\frac{Ma_i}{(iX)^3}\right)^3\right) \end{cases} \quad (4)$$
  and $s_{n+1} = (n+1)X$.

- The tree is a fork with $n+1$ internal nodes; node 0 is the root of the tree and node i is a child of node 0, for $1 \leq i \leq n$. Each internal node i has a unique leaf child with r_i requests, where $r_0 = (n+1)X - S$ and $r_i = s_i + a_i$ for $1 \leq i \leq n$.

- The bound on power consumption is $B = (n+1)^3X^3 + \sum_{i=1}^{n}\left((iX)^3 + Ma_i\right) - MS + 0.5$.

Note that for all i, $\tilde{s}_i$ is a rational number polynomial in the size of the instance. The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. Let us further define a new set of speeds:

$$\text{for } 1 \leq i \leq n, \quad s'_i = \left((iX)^3 + Ma_i\right)^{1/3}. \quad (5)$$

The speeds s'_i are not part of the instance but they will be used for the proof. Intuitively, $\tilde{s}_i$ is a close approximation of s'_i for $1 \leq i \leq n$.

The intuition behind the proof is that there is a server on each node, hence the proof works for both problems (Discrete and Discrete-Placed). Furthermore, the root node will run at speed s_{n+1}, and it will process exactly $(n+1)X$ requests. Therefore, S requests can come from its children. Because of the speed configurations, server i has exactly two choices, either speed s_i or speed $\tilde{s}_i$. The constants X and M ensure that no other speed can be used, because speeds are spaced apart enough. Either server i processes all its requests, and it must run at speed $\tilde{s}_i$, or a_i requests are forwarded to the root server and server i can then run at speed s_i. There can be at most S requests going up (server capacity limit at the root node), and because of the bound on the power consumption, there should be at least S requests going up, hence the 2-partition. Indeed, the power saved when moving from speed s'_i (and hence almost $\tilde{s}_i$) to speed s_i is linear in a_i: $s_i'^3 - s_i^3 = Ma_i$.

First we prove some preliminary properties on the $\tilde{s}_i$'s to ensure that the s'_i's are a good approximation and that speeds are spaced apart enough so that each server is restricted to its set of two speeds.

Lemma 1. *The following properties hold true:*

1. *for* $1 \leq i \leq n$, $s'_i \leq \tilde{s}_i$;
2. *for* $1 \leq i \leq n$, $s_i + a_i = iX + a_i \leq \tilde{s}_i$;
3. *for* $1 \leq i \leq n$, $\tilde{s}_i \leq (i+1)X = s_{i+1}$;
4. $\sum_{i=1}^{n} s_i'^3 \leq \sum_{i=1}^{n} \tilde{s}_i^3 \leq \sum_{i=1}^{n} s_i'^3 + 0.5$;
5. *for* $1 \leq i \leq n$, $\tilde{s}_i \leq iX + (Ma_i)^{1/3}$.

Proof. Consider the following functions on $[0,1]$: $f_1 : x \mapsto (1+x)^{1/3}$ and $f_2 : x \mapsto 1 + x/3 - x^2/9 + 5x^3/81$. We have $s'_i = iXf_1\left(\frac{Ma_i}{(iX)^3}\right)$ and $\tilde{s}_i = iXf_2\left(\frac{Ma_i}{(iX)^3}\right)$, and note that $\frac{Ma_i}{(iX)^3} \leq \frac{MS}{X^3} < 1$, because $M = 7n^2X^2$ and $X > 7n^2S$.

We prove the five properties:

1. $(\tilde{s}_i - s'_i \geq 0)$ We can verify, using several differentiations (or a computer algebra software), that $f_2(x) - f_1(x) > 0$ for $x \in [0,1]$.

2. $(iX + a_i \leq \tilde{s}_i)$ Thanks to Property 1, $\tilde{s}_i > s'_i$, and showing that $s'_i \geq s_i + a_i$ gives the result:
   $$\begin{aligned} & s'_i & \geq s_i + a_i \\ \Leftrightarrow\ & (s_i^3 + Ma_i)^{1/3} & \geq s_i + a_i \\ \Leftrightarrow\ & s_i^3 + Ma_i & \geq s_i^3 + 3s_i^2a_i + 3s_ia_i^2 + a_i^3 \\ \Leftrightarrow\ & M & \geq 3(iX)^2 + 3iXa_i + a_i^2 \end{aligned}$$
   Because $X > a_i$ and $n \geq i \geq 1$, we have $M = 7n^2X^2 \geq 3(iX)^2 + 3iXa_i + a_i^2$, hence the result.

3. $(\tilde{s}_i \leq (i+1)X)$ We have already seen that $\frac{Ma_i}{X^3} < 1$, and therefore $\tilde{s}_i \leq iX\left(1 + \frac{1}{3i} + \frac{5}{81i}\right) = \left(i + \frac{32}{81}\right)X \leq (i+1)X$.

4. $(\sum_{i=1}^{n} s_i'^3 \leq \sum_{i=1}^{n} \tilde{s}_i^3 \leq \sum_{i=1}^{n} s_i'^3 + 0.5)$ We can verify, using several differentiations (or a computer algebra software), that $\forall x \in [0,1]$,
   $$f_2(x)^3 - f_1(x)^3 \leq x^4.$$
   Then,
   $$\begin{aligned} \tilde{s}_i^3 - s_i'^3 &= (iX)^3\left(f_2\left(\frac{Ma_i}{(iX)^3}\right)^3 - f_1\left(\frac{Ma_i}{(iX)^3}\right)^3\right) \\ &\leq \frac{\left(7n^2a_i\right)^4}{i^9X}. \end{aligned}$$

Hence:

$$\begin{aligned}\sum_{i=1}^{n}\tilde{s}_i^3 &\leq \sum_{i=1}^{n}\left(s_i^{'3}+\frac{\left(7n^2a_i\right)^4}{i^9X}\right)\\ &\leq \sum_{i=1}^{n}s_i^{'3}+n\frac{\left(7n^2a_i\right)^4}{X}\\ &\leq \sum_{i=1}^{n}s_i^{'3}+n\frac{\left(7n^2S\right)^4}{X}\\ &\leq \sum_{i=1}^{n}s_i^{'3}+0.5\end{aligned}$$

The last inequality is established because $X \geq 2n\left(7n^2S\right)^4$.

5. $(\tilde{s}_i \leq iX+\frac{Ma_i}{3})$ We have $\tilde{s}_i - iX - (Ma_i)^{1/3} = \frac{Ma_i}{3(iX)^2} - (Ma_i)^{1/3} - \frac{iX}{9}\left(\frac{Ma_i}{(iX)^3}\right)^2 + \frac{5iX}{81}\left(\frac{Ma_i}{(iX)^3}\right)^3$. Then, let us first show that $-\frac{iX}{9}\left(\frac{Ma_i}{(iX)^3}\right)^2 + \frac{5iX}{81}\left(\frac{Ma_i}{(iX)^3}\right)^3 < 0$:

$$\begin{aligned}\frac{iX}{9}\left(\frac{Ma_i}{(iX)^3}\right)^2 &> \frac{5iX}{81}\left(\frac{Ma_i}{(iX)^3}\right)^3\\ \frac{9}{5}\left(iX\right)^3 &> Ma_i\\ \frac{9}{5} &> \frac{7n^2a_i}{i^3X}\end{aligned}$$

This last equation is always true since $X > 7n^2a_i$. Then let us show that $\frac{Ma_i}{3(iX)^2} - (Ma_i)^{1/3} < 0$:

$$\begin{aligned}(Ma_i)^{1/3} &> \frac{Ma_i}{3(iX)^2}\\ 3i^2X^2 &> (7n^2X^2a_i)^{2/3}\\ X^{1/3} &> \frac{7^{2/3}n^{4/3}a_i}{3i^2}\\ X &> \frac{49n^4a_i^3}{27i^6}\end{aligned}$$

which is true since $X = 2n\left(7n^2S\right)^4$.

This concludes the proof of Lemma 1. □

With these properties, we are now ready to show the NP-hardness of the problem. Note that $s_1 \leq \tilde{s_1} \leq s_2 \leq \tilde{s_2} \leq \cdots \leq s_n \leq \tilde{s_n} \leq s_{n+1}$.

Suppose first that instance $\mathcal{I}_1$ has a solution I. We assign the servers as follows: the root node is assigned a server at speed s_{n+1}; for all $i \in I$, we assign to node i a server at speed s_i, and for all $i \notin I$, we assign to node i a server at speed $\tilde{s}_i$.

Because $\tilde{s}_i \geq s_i + a_i$ (Lemma 1), the server on node i can process all its requests, for $i \notin I$. On the contrary, for $i \in I$, the server can only process s_i requests, and therefore a_i requests are going up in the tree. Hence, a total of $\sum_{i\in I} a_i = S$ requests need to be processed by the root node, in addition to the $(n+1)X - S$ requests that arrive directly to this node (and the speed is s_{n+1} so the root server can process all its requests).

Finally we need to compute the power consumption, that is:

$$\begin{aligned}E &= (n+1)^3X^3 + \sum_{i\notin I}\tilde{s}_i^3 + \sum_{i\in I}s_i^3\\ &= (n+1)^3X^3 + \sum_{i\notin I}\tilde{s}_i^3 + \sum_{i\in I}(s_i^{'3} - Ma_i)\\ &\leq (n+1)^3X^3 + \sum_{i\notin I}\tilde{s}_i^3 + \sum_{i\in I}\tilde{s}_i^3 - M\sum_{i\in I}a_i\\ &= (n+1)^3X^3 + \sum_{i=1}^{n}\tilde{s}_i^3 - MS\\ &\leq (n+1)^3X^3 + \sum_{i=1}^{n}s_i^{'3} + 0.5 - MS = B\end{aligned}$$

The power bound is respected, and therefore we have a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution. We define the set I of nodes as follows:

$$\begin{aligned}i \in I \iff &\text{there is a server on node } i\\ &\text{operating at speed } s < \tilde{s}_i\end{aligned}$$

We show that I is a solution to $\mathcal{I}_1$. First let us show that for $i \in I$, the speed of the server on node i is exactly s_i. We prove this by contradiction: if this server has a lower speed s, then the number of requests from node i that goes up to the root node is at least

$$\begin{aligned}iX + a_i - s &\geq iX + a_i - ((i-1)X + (Ma_{i-1})^{1/3})\\ &\geq X + a_i - (Ma_{i-1})^{1/3},\end{aligned}$$

because $s \leq \tilde{s}_{i-1}$, and $\tilde{s}_{i-1} \leq (i-1)X + (Ma_{i-1})^{1/3}$ (Lemma 1). This would lead to execute at the root more requests than the maximum $(n+1)X$. Indeed,

$$\begin{aligned}&(n+1)X - S + X + a_i - (Ma_{i-1})^{1/3} > (n+1)X\\ &\Leftrightarrow X > (Ma_{i-1})^{1/3} + S - a_i\\ &\Leftrightarrow X > (a_{i-1}7n^2X^2)^{1/3} + S - a_i\end{aligned}$$

We know that $S < X^{2/3}$, and that $X^{1/3} > (1 + (7n^2)^{1/3})$ (we use $a_i < S$), hence the contradiction. Therefore we conclude that for all $i \in I$, the server on node i works at speed s_i.

Then for all $i \notin I$, there is a server on node i (otherwise the root node could not process the $iX + a_i$ upcoming requests with the largest

speed $(n+1)X$), and its speed is at least $\tilde{s}_i$ by definition of I. We have seen that $\tilde{s}_i \geq iX + a_i$, so every request can be processed directly on node i.

Consequently, the power consumption is at least $E \geq (n+1)^3X^3 + \sum_{i \notin I} \tilde{s}_i^3 + \sum_{i \in I} s_i^3 \geq (n+1)^3X^3 + \sum_{i=1}^{n} s'_i - M \sum_{i \in I} a_i$ Then we have, $M \sum_{i \in I} a_i \geq MS - 0.5$ because $E \leq B$ (we have a solution of the problem). Because M and S are integers, it implies that $\sum_{i \in I} a_i \geq S$. Because of the constraint on the maximum speed at the root node, the quantity of requests coming from other nodes and processed at the root cannot be greater than S, and we have $\sum_{i \in I} a_i \leq S$. Indeed, for all $i \in I$, no more than iX requests can be executed on node i, and therefore there are at least a_i requests from node i executed at the root.

Finally, we conclude that $\sum_{i \in I} a_i = S$, and therefore that $\mathcal{I}_1$ has a solution. This concludes the proof for DISCRETE. The same instance can be used to prove the NP-completeness of DISCRETE-PLACED (which belongs to NP since DISCRETE does), because each internal node is equipped with a server in the solution. □

4.2. Mixed integer linear program

Theorem 2. *The following MILP characterizes the* DISCRETE *problem, where the unknown variables are the $x_{j,k}$'s (Boolean variables) and the $y_{i,j}$'s (rational variables), for $j \in \mathcal{N}$, $1 \leq k \leq K$ and $i \in \mathcal{C}$:*

Minimize $\sum_{j \in \mathcal{N}} \sum_{1 \leq k \leq K} x_{j,k}(\mathcal{P}_{static} + s_k^3)$
subject to

$$\text{(i)} \ \sum_{j \in \mathcal{N}} y_{i,j} = r_i, \qquad i \in \mathcal{C}$$
$$\text{(ii)} \ \sum_{1 \leq k \leq K} x_{j,k} \leq 1, \qquad j \in \mathcal{N}$$
$$\text{(iii)} \ \sum_{i \in \mathcal{C}} y_{i,j} \leq \sum_{1 \leq k \leq K} x_{j,k} s_k, \qquad j \in \mathcal{N}$$

Proof. The constants are the r_i's for $i \in C$, and the s_k's for $1 \leq k \leq K$, and we consider the following variables:

- $x_{j,k}$ is a boolean variable equal to 1 if j is a server operated at speed s_k, for $j \in \mathcal{N}$ and $1 \leq k \leq K$; $x_{j,k} = 0$ otherwise.
- $y_{i,j}$ is a rational variable equal to $r_{i,j}$, the number of requests of client $i \in \mathcal{C}$ processed by server $j \in \mathcal{N}$; if j is not an ancestor of i in the tree, we directly set $y_{i,j} = 0$.

Then the constraints are:

- For all $i \in \mathcal{C}$, all requests of client i must be processed: $\forall i \in \mathcal{C}, \sum_{j \in \mathcal{N}} y_{i,j} = r_i$;
- Each server is assigned at most one speed: $\forall j \in \mathcal{N}, \sum_{1 \leq k \leq K} x_{j,k} \leq 1$; note that a node j is equipped with a server if and only if $\sum_{1 \leq k \leq K} x_{j,k} = 1$;
- The processing capacity of any server cannot be exceeded: $\forall j \in \mathcal{N}, \sum_{i \in \mathcal{C}} y_{i,j} \leq \sum_{1 \leq k \leq K} x_{j,k} s_k$.

Finally, we minimize the total power consumption. Overall, there are $|\mathcal{C}| + 2|\mathcal{N}|$ constraints and $|\mathcal{N}| \cdot (|K| + |\mathcal{C}|)$ variables in this MILP. □

5. Heuristics

In this section, we propose some polynomial-time heuristics for the DISCRETE problem. We start by outlining the general principles that have guided their design before exposing the details for each heuristic.

5.1. General principles

As already mentioned in Section 1, the heuristics work in two steps: (i) server placement and (ii) request assignment. The placement step of the heuristics relies on the following result:

Proposition 1. *Given a fixed set of servers deployed on a tree of size $s = |\mathcal{C}| + |\mathcal{N}|$ and assuming continuous speeds, the optimal assignment* ALG-CONT-PLACED *of requests to servers that uses all these servers and minimizes power consumption can be determined in time $O(s^2)$ (see Algorithm 1).*

The placement step works incrementally: to compute a solution with k servers, the heuristic starts from a solution with $k-1$ servers, and then greedily tests the addition of one additional server. It uses Proposition 1 to compute the optimal assignment of requests with this additional server, computes the corresponding power, and returns the best solution over all possible choices for the additional server.

In order to prove Proposition 1, first let us define formally the problem with continuous speeds and already placed servers that we are solving:

Definition 1 (CONTINUOUS-PLACED)**.** Given a distribution tree (with a number of requests per client), with servers turned on and already placed on the tree, decide how to distribute client requests among them (which can also be seen as assigning the speed, equal to the number of requests, of each server), in order to minimize the total power consumption.

Note that in the following, because placed servers are all assumed turned on, we do not need to account for $\mathcal{P}_{\text{static}}$.

Definition 2. Let T be a distribution tree. Let N and N' be two nodes of T:

- N' is a *server-child* of N if there is a path from N' to N (i.e., N is an ancestor of N' in the tree), and there is a server on N'. Furthermore, we call it a *direct server-child* if it

is a server-child and there are no other servers on the path from N' to N. Symmetrically, N is a server-parent of N'.

- Given a solution where N' is a child of N, and such that there is a server on N' that computes r' requests, and a server on N that computes r requests, then a *transfer of* $t \in [0, r']$ *requests* from N' to N is a solution where N' computes $r' - t$ requests and N computes $r + t$ requests.

Given a solution to CONTINUOUS-PLACED we consider the two following hypothesis:

Hypothesis 1. The number of requests processed by a server is never smaller than the number of requests processed by each of its server-children.

Hypothesis 2. If a server does not process all requests that are available in its subtree (i.e., some of these requests are processed higher in the tree), then the number of requests that it processes is equal to the number of requests processed by its direct server-parent.

Lemma 2. *There exists an optimal solution to* CONTINUOUS-PLACED *that satisfies hypothesis 1.*

Proof. Consider an optimal solution to CONTINUOUS-PLACED that does not satisfy hypothesis 1. Then there exists a server N_1 processing w_1 requests, while one of its server-children N_2 is processing $w_2 > w_1$ requests. By transferring $w_2 - w_1$ requests from N_2 to N_1 (which is possible because N_1 is a parent of N_2), we can construct a solution such that N_1 executes w_2 requests and N_2 executes w_1 requests. This solution has the same power consumption as the previous one.

Similarly we can do this in the new solution, on all nodes that violate hypothesis 1, until the solution does not violate anymore hypothesis 1. □

Lemma 3. *If a solution to* CONTINUOUS-PLACED *is optimal, then it satisfies hypothesis 2.*

Proof. Let us consider a solution that does not satisfy hypothesis 2. Then there exists a server N_2 processing w_2 requests, and its direct parent-server N_1 is processing $w_1 \neq w_2$ requests. Moreover, N_2 leaves l requests to be processed higher in the tree (either by N_1 or by an ancestor of N_1).

1. if $w_1 < w_2$, then transferring $\frac{w_2 - w_1}{2}$ additional requests from N_2 to N_1 is still a valid solution since N_1 is a parent of N_2, and the power consumption of this solution is better $(2\left(\frac{w_1+w_2}{2}\right)^3 < w_1^3 + w_2^3)$.
2. if $w_2 < w_1$, then it would be better to process $l' = \min\left(l, \frac{w_1 - w_2}{2}\right)$ more requests at node N_2. This is possible because these requests are processed higher in the tree and we can always exchange them with some requests of N_1 if N_1 is not processing them. In the new solution, N_2 processes $w_2 + l'$ requests, while N_1 processes $w_1 - l'$ requests. Because $l' > 0$, and by convexity, this new solution has a better power consumption.

Hence the solution is not optimal, and all optimal solutions satisfy hypothesis 2. □

Theorem 3. *If a solution to* CONTINUOUS-PLACED *satisfies hypothesis 1 and 2, then it is optimal.*

To prove this theorem, let us show that

1. there is a unique solution that satisfies both hypotheses;
2. this solution is optimal.

Proposition 2. *Given two optimal solutions to* CONTINUOUS-PLACED *that satisfy hypothesis 1 and 2, consider a server N that computes w_1 (resp. w_2) requests in the first (resp. second) solution, and that leaves t_1 (resp. t_2) requests to be processed higher in the tree.*

- *If $w_1 > w_2$ and $t_1 \geq t_2$, then there exists a direct server-child N' of N that computes w_1' (resp. w_2') requests in the first (resp. second) solution, and that transfers t_1' (resp. t_2') requests to N, such that $w_1' > w_2'$ and $t_1' > t_2'$.*
- *If $w_1 \geq w_2$ and $t_1 > t_2$, then there exists a direct server-child N' of N that computes w_1' (resp. w_2') requests in the first (resp. second) solution, and that transfers t_1' (resp. t_2') requests to N, such that $w_1' \geq w_2'$ and $t_1' > t_2'$.*

Proof. In order to prove this proposition, note that we know that $w_i + t_i$ is equal to the sum of the requests coming directly into N, and of all requests transferred from N's server-children. Then, because $w_1 + t_1 > w_2 + t_2$ in both cases, there exists a direct server-child of N such that $t_1' > t_2'$. Then necessarily $t_1' > 0$, and with hypothesis 2, $w_1' = w_1$. Finally, with hypothesis 1, $w_2' \leq w_2$, hence the results: if $w_2 < w_1$, then $w_2' \leq w_2 < w_1 = w_1'$, and if $w_2 \leq w_1$, then $w_2' \leq w_2 \leq w_1 = w_1'$. □

Proof of Theorem 3. **Uniqueness of the optimal solution:** To show that there is a unique optimal solution, we proceed by induction. If there is a unique server, then there is a unique solution where this server processes all requests.

Let us now consider a tree with n servers. Let us consider two optimal solutions that satisfy hypothesis 1 and 2. Finally, let us consider w_1 (resp. w_2) the number of requests computed by the server at the root of the tree in the first (resp. second) solution. The transfer at the root of the tree is necessary null, i.e., $t_1 = t_2 = 0$ at the root. Note that if there is no server at the root of the tree, then we can divide the tree into subtrees with a server at their roots, and by induction the two solutions are identical. If there is only one subtree with a server at its root, then we conduct the same reasoning on this subtree, and the transfer is necessary null because there are no servers upper in the tree. N denotes the server at the root of the tree.

Then, there are five different possibilities:

- If $w_1 > w_2$, with Proposition 2, we can construct an infinite sequence of servers s starting from N such that $w_1^{(s)} > w_2^{(s)}$ and $t_1^{(s)} \geq t_2^{(s)}$, because initially $t_1 = t_2$ (no transfer from the root node). Hence, we have a contradiction since there are only n servers.
- The case with $w_2 > w_1$ is symmetrical.
- If $w_1 = w_2$, and if there exists a direct server-child s of N such that $t_1^{(s)} > t_2^{(s)}$, then with hypothesis 2, $w_1^{(s)} = w_1$, and with hypothesis 1, $w_2^{(s)} \leq w_2$. Finally, with Proposition 2, we can construct an infinite sequence of servers $\tilde{s}$ starting from s such that $w_1^{(\tilde{s})} \geq w_2^{(\tilde{s})}$ and $t_1^{(\tilde{s})} > t_2^{(\tilde{s})}$. Hence we have a contradiction since there are only n servers.
- The case with $w_1 = w_2$, and there exists a direct server-child s of N such that $t_1^{(s)} < t_2^{(s)}$, is symmetrical.
- Finally, if $w_1 = w_2$, and for all direct server-child s of N, $t_1^{(s)} = t_2^{(s)}$, then we look at the sub-problem for all sub-trees of N, and by induction hypothesis, the two solutions are equal.

Finally, we have shown that two solutions to Continuous-Placed that satisfy both hypothesis are identical.

Existence of a solution: With Lemma 2, we know that there exists an optimal solution to Continuous-Placed that satisfies hypothesis 1. With Lemma 3, we know that this solution satisfies hypothesis 2. Therefore, it is exactly the solution that satisfies both hypothesis.

Finally, we have shown that if a solution satisfies both hypothesis, it is optimal. □

We now introduce the algorithm Alg-Cont-Placed that computes a distribution of the requests over a set of placed servers. First, note that if there are requests coming in a node such that there are no servers placed on that node or on any of the ancestors of that node, then there is no solution to the problem Continuous-Placed: these requests cannot be satisfied.

Definition 3 (server-subtrees)**.** For a given node N of a tree T, a *server-subtree* of N is a subtree of T below N such that

1. there is a server at the root N' of the subtree;
2. there are no servers on any nodes of the path of T from N to N'.

Hence we can consider trees such that there is a server on the root node (otherwise, either there is no solution to Continuous-Placed, or we can divide a tree into the disjoint server-subtrees of the root of the original tree).

Theorem 4. *The solution returned by Algorithm 1 satisfies hypothesis 1 and 2, therefore it is optimal.*

Proof. It is easy to verify that both hypothesis are verified by the result of the algorithm. Given a node N,

- They are verified at the beginning of the execution of the algorithm for all server subtrees of N.
- At all time during the execution of the "while" loop, they are verified for all the servers below N.
- The "while" loop ensures that both hypothesis are verified for N at the end of the execution.

□

Note that the proofs also work if the only set of speeds allowed is $[s_{\min}, s_{\max}]$. We detail in Algorithm 2 how to modify Alg-Cont-Placed to account for this restriction.

The placement step assumes continuous speeds, hence the loads assigned by Alg-Cont-Placed to each server do not take the set of actual speeds into account. The second step of the heuristics consists in determining a discrete speed for each server, which usually leads to re-assigning some requests, as explained in Section 1. While the first step of the heuristics is common to all heuristics, we outline below three different methods to perform this request assignment step.

Algorithm 1: Procedure ALG-CONT-PLACED

```
1  procedure ALG-CONT-PLACED(T)
2  begin
3      return sub-ALG-CONT-PLACED (T, root)
4
5  procedure sub-ALG-CONT-PLACED(T, N) /* here N is a node of T */
6  begin
7      for s ∈ Child(N) do
8          T_s = sub-ALG-CONT-PLACED (T, s);
9      Let R be the number of requests that are sent by clients on internal nodes that are not in server-subtrees of N;
10     Let S_N = R be the speed of the server on node N;
11     while there exists a server-child of N with a speed greater than S_N do
12         Let S_max be the maximum speed of a server-child of N;
13         Let S_2 be the second maximum speed of a server-child of N (S_2 ≠ S_max), S_2 = 0 if there is no second maximum speed;
14         Let N_1, ..., N_k be the k server-children of N that work at speed S_max;
15         Let S' = (kS_max + R)/(k+1);
16         if S' > S_2 then
17             N_1, ..., N_k now work at speed S', and transfer S_max − S' additional requests to N;
18             R ← S' /* We will exit the while loop at the next step */
19         else
20             N_1, ..., N_k now work at speed S_2, and transfer S_max − S_2 additional requests to N;
21             R ← R + k(S_max − S_2) /* Note that R < S_2 */
22     return T
```

5.2. List of heuristics

We provide here three different heuristics to determine the actual speed of each server. In the first heuristic, GREEDY, we assign the smallest speed equal to or greater than the load given by ALG-CONT-PLACED to each server.

A first remark is that if there is no speed greater than the value determined by ALG-CONT-PLACED for some server, then there does not exist a solution for this (given) placement.

Algorithm 2: Procedure ALG-CONT-PLACED where allowed speeds range from $S_{\min}$ to $S_{\max}$.

```
1  procedure ALG-CONT-PLACED-BOUNDED(T)
2  begin
3      Let T = sub-ALG-CONT-PLACED-BOUNDED (T, root);
4      Let S be the speed of the root of T;
5      if S > S_max then
6          return There are no solutions
7      else
8          return T
9
10 procedure sub-ALG-CONT-PLACED-BOUNDED(T, N)
11 begin
12     for s ∈ Child(N) do
13         T_s = sub-ALG-CONT-PLACED-BOUNDED (T, s);
14     Let R be the number of requests that are sent by clients on internal nodes that are not in server-subtrees of N;
15     Let S_N = max(R, S_min) be the speed of the server on node N;
16     while there exists a server-child of N with a speed greater than S_N do
17         Let S_max be the maximum speed of a server-child of N;
18         Let S_2 be the second maximum speed of a server-child of N (S_2 ≠ S_max), S_2 = S_min if there is no second maximum speed;
19         Let N_1, ..., N_k be the k server-children of N that work at speed S_max;
20         Let S' = (kS_max + R)/(k+1);
21         if S' > S_2 then
22             N_1, ..., N_k now work at speed S', and transfer S_max − S' additional requests to N;
23             R ← S' /* We will exit the while loop at the next step */
24         else
25             N_1, ..., N_k now work at speed S_2, and transfer S_max − S_2 additional requests to N;
26             R ← R + k(S_max − S_2) /* Note that R < S_2 */
27     return T
```

Theorem 5. *Given a distribution tree (with a number of requests per client), with servers turned on and already placed on the tree, if the maximum speed given by the solution of* Alg-Cont-Placed *is greater than the maximum discrete speed, then there is no solution.*

Proof. Assume a distribution tree with servers turned on and already placed on the tree. Let us call $\mathcal{S}$ the solution given by Alg-Cont-Placed, such that one server works at a speed s greater than the maximum discrete speed (s_K). Since the result given by Alg-Cont-Placed follows hypothesis 1 (Theorem 4), then in particular the server placed at the root of the tree works at speed s.

Let us assume that there exists a solution $\mathcal{S}'$ to Discrete-Placed. Then the root of the tree is processed at a speed lower than or equal to s_K. Let us consider the $s - s_K$ requests that are processed on the root in $\mathcal{S}$, but are not processed on the root in $\mathcal{S}'$. Then, in $\mathcal{S}'$ they are processed on a server below the root:

- Either this server works at speed s in $\mathcal{S}$, but then we can consider the subtree rooted in this server and reach a contradiction;
- Either this server works at a speed lower than s in $\mathcal{S}$, and because $\mathcal{S}$ follows hypothesis 2 (Theorem 4), it does not transfer any requests higher the tree. Hence necessarily, it cannot process some of the $s - s_K$ requests that are not processed on the root in $\mathcal{S}'$.

Finally, necessarily, the $s - s_K$ requests that are not processed on the root are processed on a node that works at speed $s > s_K$ in $\mathcal{S}$, and we have a contradiction.

In conclusion, if the maximum speed given by the solution of Alg-Cont-Placed is greater than the maximum discrete speed, there is no solution to Discrete-Placed. □

Furthermore, we state an important result for the Greedy heuristic:

Theorem 6. *Given a distribution tree (with a number of requests per client), with servers turned on and already placed on the tree,* Greedy *is a* $(1 + \frac{\max_{1 \leq i < K}(s_{i+1} - s_i)}{s_1})^3$*-approximation to the* Discrete-Placed *problem.*

Proof. First, note that the optimal solution to Continuous-Placed, where the set of speeds is bounded in the interval $[s_1, s_K]$, is a lower-bound on the optimal solution to Discrete-Placed. Indeed, any solution to Discrete-Placed is a solution to Continuous-Placed.

Then, for each server N, let us call $s_N^{\{c\}}$ the speed given by the Alg-Cont-Placed-Bounded with bounded speeds (Algorithm 2), and $s_N^{\{g\}}$ the speed given by the Greedy algorithm. There exists i such that $s_N^{\{g\}} = s_{i+1}$ (with $0 \leq i < K$ and $s_0 = 0$), and by definition of $s_N^{\{g\}}$,

$$s_N^{\{g\}} \geq s_N^{\{c\}} > s_i = s_N^{\{g\}} - (s_{i+1} - s_i)\,.$$

Furthermore, if $s_N^{\{g\}} = s_1$, then $s_N^{\{c\}} = s_N^{\{g\}}$. Therefore, if $i > 1$, then $s_N^{\{g\}} < s_N^{\{c\}} \left(1 + \frac{s_{i+1} - s_i}{s_N^{\{c\}}}\right)$, and in all cases,

$$s_N^{\{g\}} < s_N^{\{c\}} \left(1 + \frac{\max_{1 \leq i < K} (s_{i+1} - s_i)}{s_1}\right).$$

Finally, we can bound the total energy consumption $E^{\{g\}}$ of the Greedy solution:

$$\begin{aligned} E^{\{g\}} &= \sum_N s_N^{\{g\}3} \\ &\leq \sum_N s_N^{\{c\}3} \left(1 + \frac{\max_{1 \leq i < K} (s_{i+1} - s_i)}{s_1}\right)^3 \\ &\leq E^{\{opt\}} \left(1 + \frac{\max_{1 \leq i < K} (s_{i+1} - s_i)}{s_1}\right)^3, \end{aligned}$$

where $E^{\{opt\}}$ is the optimal energy consumption. Indeed, $\sum_N s_N^{\{c\}3}$ corresponds to the lower bound of the optimal energy consumption in the continuous case. This concludes the proof. □

The next two heuristics, Speed and Excess, improve the Greedy heuristic by trying to modify the load of each server, via request reassignment. The goal is to decrease the speed of some servers. More precisely, in the procedure, which is called Equilibrate and detailed in Algorithm 3, if a server is not loaded up to its full capacity (meaning its load is equal to its capacity), then the heuristics take some load out of its children until this server reaches its capacity (see the loop line 15). The capacity of a server is defined as the maximum between its actual speed and the maximum speed of its children (see line 12), hence transferring even more load to this server if one of its children has a higher speed (and thus we should be able to reduce the speed of at least one child). This may happen if we have decreased the speed of the current node in a previous step of the algorithm, but not the speed of its children.

The main difference between the two heuristics Speed and Excess lies in the selection of the children whose load is taken out:

- In the SPEED heuristic, we favor the children whose servers have the largest speeds. To break ties if two children of a given server have the same speed, we favor the one with the smallest load. The idea is that the gain in power will be more important if we can decrease the execution speed of a server with a large speed (favor large speeds); and if there is a tie, there is more chance to decrease the speed of a server if its load is small.
- On the contrary, in the EXCESS heuristic, we favor children with small *excess*. The excess of a server is defined as the difference between its load and the largest speed below it. The idea is that we will be able to decrease the speed of more servers if we favor small excess. Finally, when two children have the same excess, we favor the one with the largest load.

Algorithm 3: Procedure EQUILIBRATE

```
1  procedure EQUILIBRATE(T)
2  begin
3      Apply the GREEDY heuristic on T;
4      return sub-EQUILIBRATE (T, root)
5
6
7  procedure sub-EQUILIBRATE(T, N) /* here
   N is a node of T */
8  begin
9      Let L the list of children of N sorted as
       favored by the heuristic;
10     Let (i_N, w_N) be the indices of the speed
       and load of N;
11     Let i_N' be the index of the maximum
       speed of the children of N;
12     if s_{i_N'} > s_{i_N} then
13         i_N ← i_N';
14     remaining ← s_{i_N} − w_N;
15     while remaining ≠ 0 do
16         Let N' be the first element of L;
17         Let (i_N', w_N') be the indices of the
           speed and load of N';
18         temp ← min(remaining, w_N' − s_{i_N'});
19         w_N' ← w_N' − temp;
20         remaining ← remaining − temp;
21         w_N ← w_N + temp;
22         Update the index of the speed of N';
23         Update the order of L to account for
           the new load and speed of N';
24     for N' ∈ children of N do
25         T ← sub-EQUILIBRATE (T, N');
26     return T
```

Recall that K is the number of speeds and $s = |\mathcal{C}| + |\mathcal{N}|$. The complexity of the GREEDY heuristic is $O(s^2)$, the most costly part being the call to ALG-CONT-PLACED. For the EQUILIBRATE procedure, and hence for the SPEED and EXCESS heuristics, the complexity becomes $O(Ks^2 \log s)$. First, it takes $s \log s$ operations to sort the children of each of the s nodes in a pre-treatment phase. Then, the procedure sub-EQUILIBRATE is called s times, and it has a cost $O(Ks \log s)$: in the worst case, we visit each child, for each possible speed, to fill the parent at full capacity, and we have to include the cost of updating the list of sorted children, that can be done in $O(\log s)$.

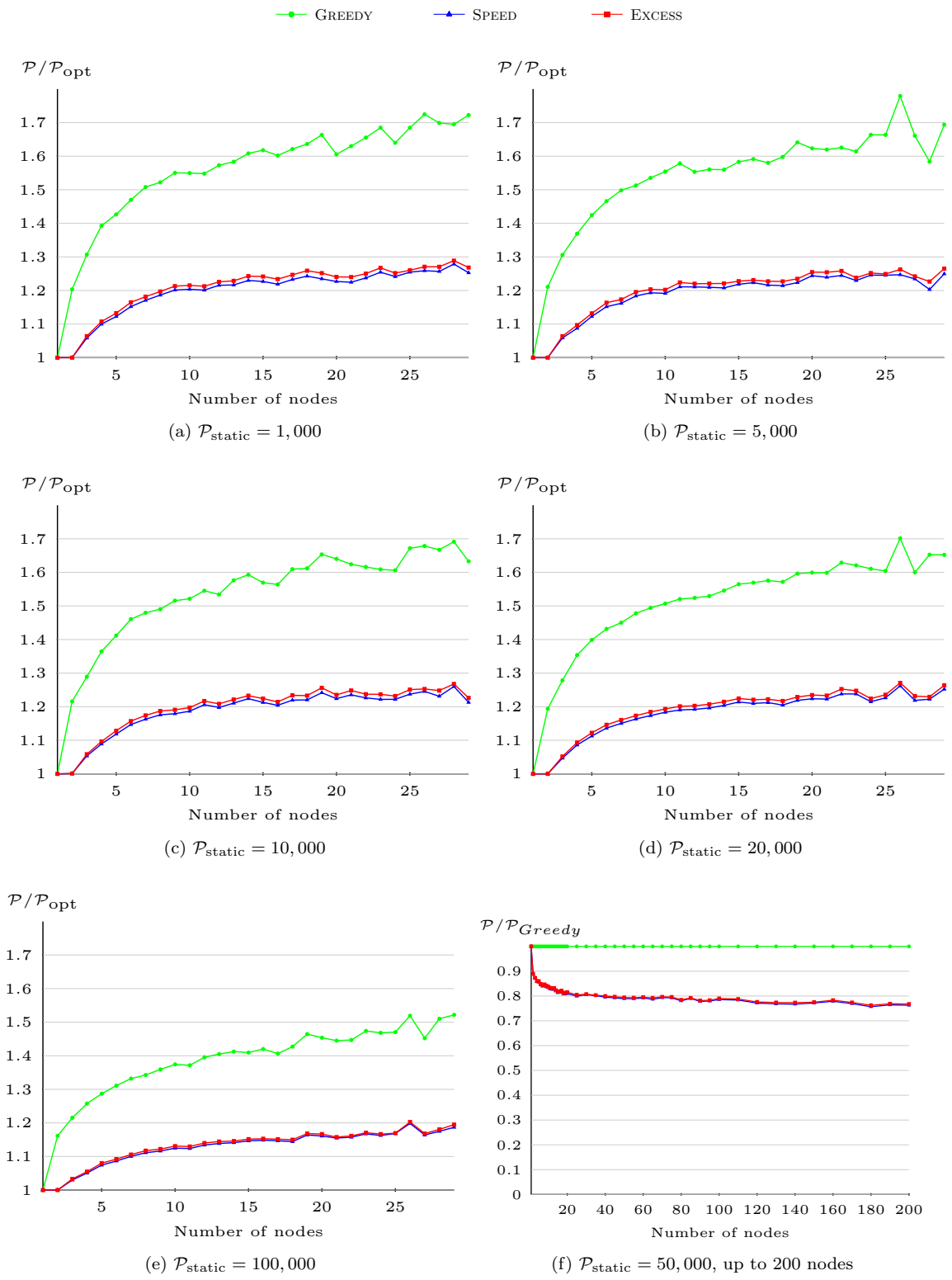
6. Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 5. All the source code, together with scripts to obtain additional results, are publicly available [17]. The heuristics have been coded using the programming language OCaml, while the MILP computing the optimal solution is generated using the C language and solved using IBM Cplex [18].

In order to evaluate the heuristics, we have generated more than 100 random trees for each simulation. To simplify the generation, each internal node in the tree has a unique client leaf, which is assigned a random rational number of requests between 0 and 100 (unless stated otherwise). For processor speeds, unless stated otherwise, we use five speeds spaced as those of the Intel Xscale, following [19, 20]: the largest speed can process 150 requests, and the ratio of the different speeds to the largest speed is then: $(0.15, 0.4, 0.6, 0.8, 1)$. In [19, 20], the static power is equal to the power consumed in the lowest speed, which here corresponds to $(0.15 * 150)^3 \approx 11,000$.

We have conducted five different sets of simulations to assess the impact of the number of nodes, of static power, of the number of available speeds, of the maximum speed, and of the total load of requests. Finally, we have studied the execution time of the heuristics.

Impact of the number of nodes. In the first set of simulations, we study the impact of the number of nodes on power consumption. In Figure 1(a-e), we plot the ratio of the power returned by the heuristics over the power of the optimal solution, with a static power of 1,000, 5,000, 10,000, 20,000 and 100,000 respectively. While one could expect that the performance of the heuristics would decrease for larger trees, it seems that SPEED and



(a) $\mathcal{P}_{\text{static}} = 1,000$

(b) $\mathcal{P}_{\text{static}} = 5,000$

(c) $\mathcal{P}_{\text{static}} = 10,000$

(d) $\mathcal{P}_{\text{static}} = 20,000$

(e) $\mathcal{P}_{\text{static}} = 100,000$

(f) $\mathcal{P}_{\text{static}} = 50,000$, up to 200 nodes

Figure 1: Study of the impact of the number of nodes, for random requests between 0 and 100, average on 100 tests.

Excess reach a plateau after approximately 15 nodes, and that on average the maximum waste is between 20 and 25%. Furthermore, when the static power is higher (100,000), this maximum waste is even below 20%. This observed plateau is very likely correlated to the set of speeds and to the static power. This plateau makes sense in practice if we assume that the first step (the placement step) of the heuristics is not too far from the optimal solutio,n because the Greedy heuristic is

an approximation algorithm. SPEED and EXCESS are just improvements of the GREEDY heuristic. It would be interesting to see how much they improve the approximation factor, though probably complicated.

Figure 1f provides results at larger scale: we plot the ratio of the power returned by SPEED and EXCESS over the power consumption of GREEDY, with a static power of 50,000, but for a larger number of nodes (up to 200 nodes). We see that SPEED and EXCESS are still consistently better than GREEDY even with a larger number of nodes, with a power consumption of around 80% of the power consumption for GREEDY.

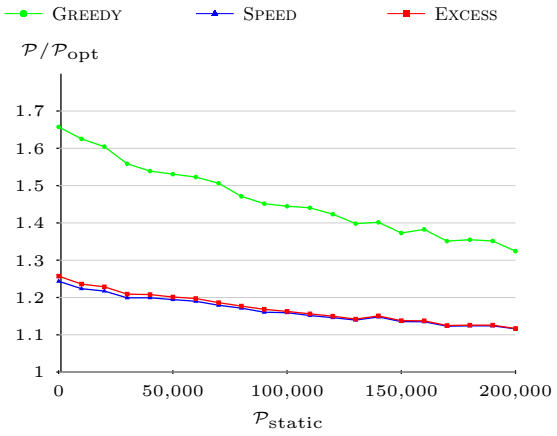


Figure 2: Study of the impact of the static power, for trees of 20 nodes, for random requests between 0 and 100, average on 100 tests.

Impact of the static power. In the second set of simulations, we have studied the impact of the static power on total power consumption. In Figure 2, we plot the ratio of the power returned by the heuristics over the power of the optimal solution, with a static power varying between 0 and 200,000 for trees of 20 nodes. Note that the higher the static power, the better the results. Indeed, at some point, what matters most is the number of servers placed, and not the allocation of requests, hence GREEDY gets closer to the optimal solution as well.

Impact of the number of speeds. In this third set of simulations, we have studied the impact of the number of speeds on power consumption. For this set of simulations, we do not use Intel speeds anymore, but instead speeds that are equally distributed between 0 and 150. In Figure 3, we plot the ratio of the power returned by the heuristics over the power of the optimal solution, with a static power of 50,000, for trees of 20 nodes, with the number of speeds varying from one (150) and ten (15, 30, 45, 60, 75, 90, 105, 120, 135, 150). When there is only one speed, obviously the results are as good as they can be and only depend on the allocation heuristic. Then starting from three speeds, we observe that the more speeds, the better the results. This was expected since the more speeds we have, the closer we can get to the optimal solution computed by ALG-CONT-PLACED, and the better the results. The fact that the results are better with two speeds than three can be explained by the fact that with only two speeds, it is still easier to find the optimal speed (hence a lower ratio than with three speeds), but a mistake is very expensive, hence a result that is not as good as with four speeds. A final remark: when speeds are equally distributed, the (proven) approximation ratio of the GREEDY heuristic is 8. However in Figure 3 we see that the ratio never goes above 1.8.

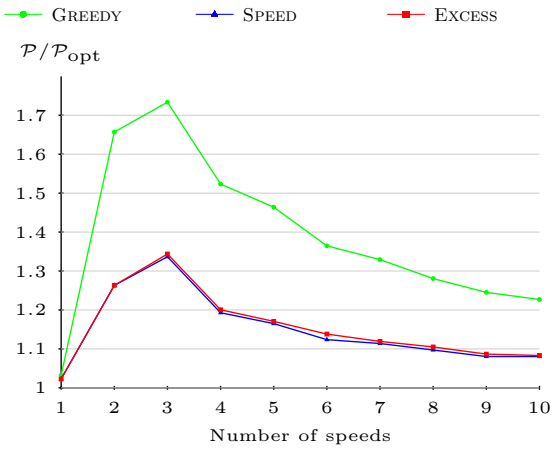


Figure 3: Study of the impact of the number of speeds, for trees of 20 nodes, static power of 50,000, for random requests between 0 and 100, average on 100 tests.

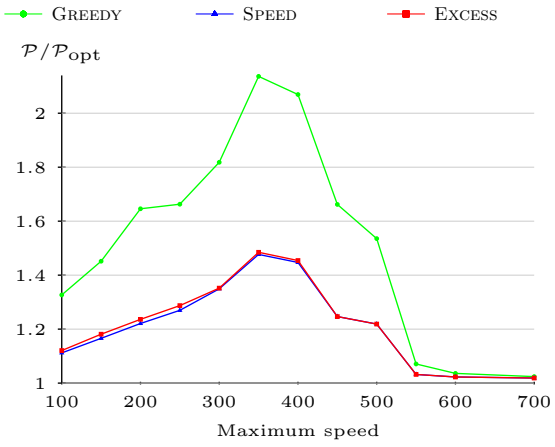


Figure 4: Study of the impact of the maximum speed, for trees of 20 nodes, static power of 10,000, for random requests between 0 and 100, average on 100 tests.

Impact of the maximum speed. In this fourth set of simulations, we have studied the impact of the maximum speed on power consumption. As for the previous set of simulations, for this set of simulations, we use speeds that are equally distributed between 0 and the maximum speed. We assume that there are always five speeds. In Figure 4, we plot the ratio of the power returned by the heuristics over the power of the optimal solution, with a static power of 10,000, for trees of 20 nodes, with the maximum speed varying from 100 and 700. When the maximum speed is low, the results are good: indeed, since the maximum speed is low, the different speeds are close, and an error in choosing the speed for a server is not too expensive. Then the ratio of the power returned by our heuristics over the optimal power consumption grows until reaching a peak (here for a maximum speed around 350): the cost of a bad choice in server speeds gets more and more costly. Finally, after reaching that peak, the ratio starts decreasing. The reason is that the difference between two consecutive speeds is so large that errors in choosing the server speeds get less and less frequent. Finally, this ratio is almost one when the maximum speed is 600, because the difference is such that (i) the smallest speed (120) is greater than the number of requests per node, and (ii) the cost to create a server at the second smallest speed (240) is almost three times the cost to put one server at the minimum speed. In that case, all solutions will put only servers at minimum speeds in order to minimize the power consumption.

Impact of the total load of the tree. Finally, in the last set of simulations, we have studied the impact of the total load (i.e., the total number of requests) of the tree on the power consumption. In Figure 5, we plot the ratio of the power returned by the heuristics over the power of the optimal solution, for trees of 20 nodes, and for a static power of 1,000, 5,000, 10,000, 20,000, 100,000 and 1,000,000 respectively. When the number of requests is uniformly chosen for all nodes between 0 and 100, the total number of requests does not impact the performance of the heuristics: the average ratio stays constant with the total load of the tree, for all values of static power. Note that the total number of requests is roughly between 800 and 1200, because of the average number of requests per node and the number of nodes (it could vary between 0 and 2000 in extreme cases, but we almost never obtain randomly such a tree).

In Tables 2 and 3, we have completed this study by changing the average number of requests per node, so that we could explore the whole range of total number of requests (between 0 and 2000). Therefore, in order to obtain an average load per node of X, the number of requests per node is a random number uniformly chosen between $X-20$ and $X+20$. For these simulations, we have chosen trees of 20 nodes with a static energy of $10,000$ for Table 2, and 1,000,000 for Table 3. The main difference in behavior between these two values of static power is when there is a very low total load. With $\mathcal{P}_{\text{static}} = 10,000$, the static power does not impact as much on the average power consumption as the dynamic power, so we need more servers with low speeds, which is harder to do. This is why the behavior is worse than with larger loads for a small $\mathcal{P}_{\text{static}}$. On the contrary, with large static power $(1,000,000)$, it is better to have as few servers as possible when the load is small, which is easier to schedule. With larger loads, the ratio is constant (within an interval of more or less 5%), which was the behavior observed in Figure 5. Finally, for total loads of 800 and 1200, the performance of the heuristics is better with the restricted set of number of requests per node than when it was uniformly distributed between 0 and 100.

Execution time of the algorithms. Finally, we study the impact of the number of nodes on the execution time. Indeed, all other parameters do not modify significantly the execution time of the heuristics. In Figure 6, we plot the execution time of all heuristics, with a static power of $50,000$. We study this execution time for trees as big as 200 nodes. The important observation is that the additional layer of Speed and Excess over the Greedy heuristic is almost free. Using a computer algebra software, we were able to approximate their execution time as $f(x) = 61.983 - 61.577\log(x) + 15.241x - 3.191x\log(x) + 0.017013x^2$.

Summary of simulation results. To conclude on the different studies, the first observation is somewhat expected: there is a huge gap between the Greedy heuristic, and the Speed and Excess heuristics: there is a degradation w.r.t. the optimal of 50 to 70% when using the Greedy heuristic with 10 to 30 nodes, while it is only approximately 20% (or less with larger static power) when using the Speed or Excess heuristic. The difference between the Speed and Excess heuristics is negligible, although it should be noted that on average, the Speed heuristic performs slightly ($\approx 1\%$) better than the Excess heuristic. Furthermore, it seems that what matters most for the competitiveness of the heuristics is the set of speeds and the static power $\mathcal{P}_{\text{static}}$. In particular, the number of speeds is very important: the closer the speeds are to each other, the better the

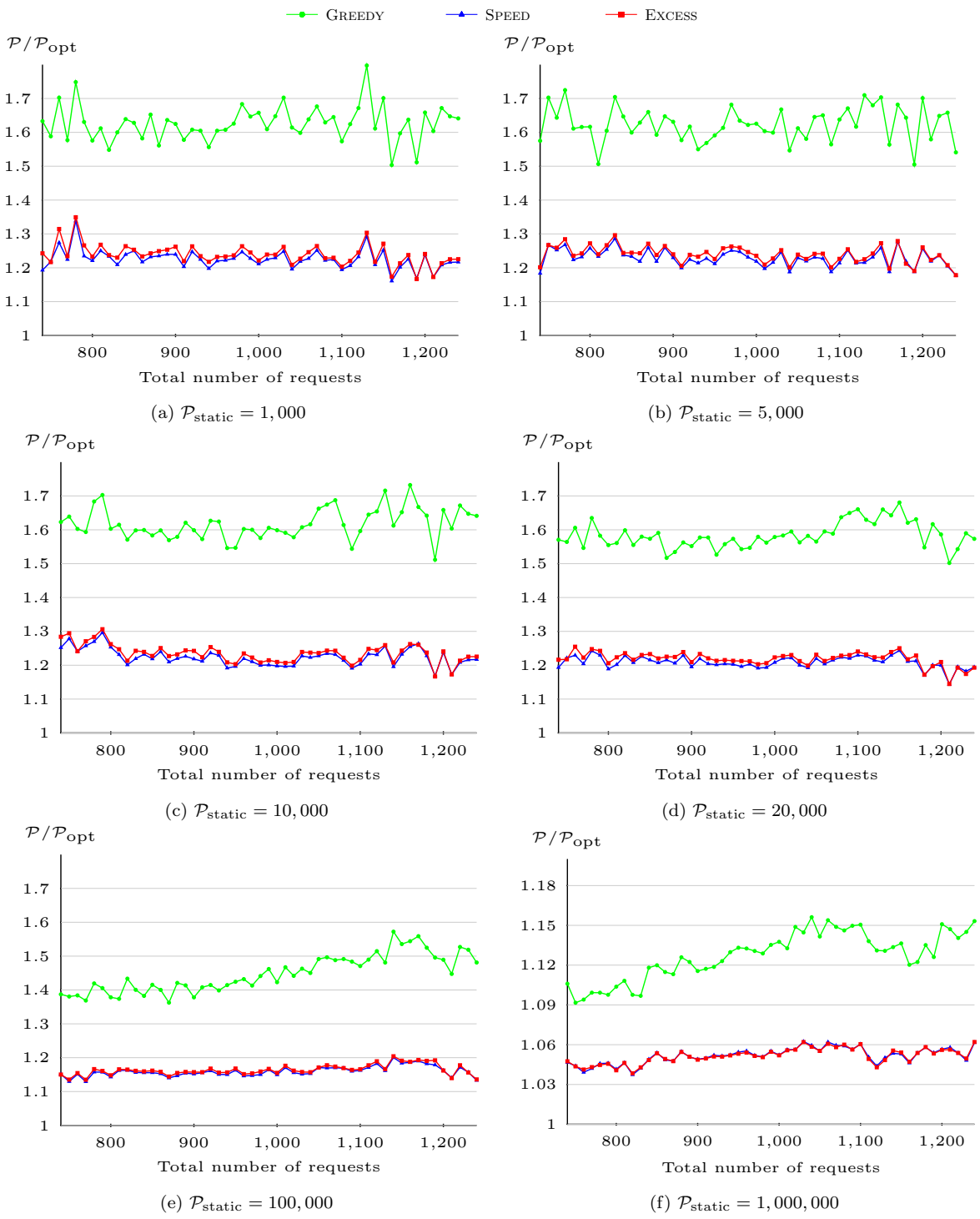


Figure 5: Study of the impact of the total load, for trees of 20 nodes, for random requests between 0 and 100, average on 100 tests.

results. Above a certain number of nodes (≈ 15), the ratio of the results of the heuristics over the optimal solution seems to reach a threshold (independently of the load and the static power), but the value of this threshold depends on the set of speeds and on the static power. Higher static power lowers the value of the threshold: at some point, what matters most is the number of servers, even if they are all at maximum speed. Similarly, the smaller the gap between two consecutive speeds, then the closer we can get to the optimal solution computed by Alg-Cont-Placed, and the better the results.

Total load	Average load per node	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (GREEDY)	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (SPEED)	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (EXCESS)
400	20	1.995011	1.427027	1.426256
800	40	1.583972	1.203851	1.206865
1200	60	1.366683	1.153863	1.155883
1600	80	1.372238	1.125676	1.128857
2000	100	1.364758	1.115750	1.114709

Table 2: Results for different total loads, $\mathcal{P}_{\text{static}} = 10,000$.

Total load	Average load per node	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (GREEDY)	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (SPEED)	$\mathcal{P}/\mathcal{P}_{\text{opt}}$ (EXCESS)
400	20	1.057985	1.016464	1.017133
800	40	1.080061	1.029879	1.029933
1200	60	1.080584	1.029221	1.028828
1600	80	1.125891	1.050705	1.051454
2000	100	1.185380	1.063981	1.063155

Table 3: Results for different total loads, $\mathcal{P}_{\text{static}} = 1,000,000$.

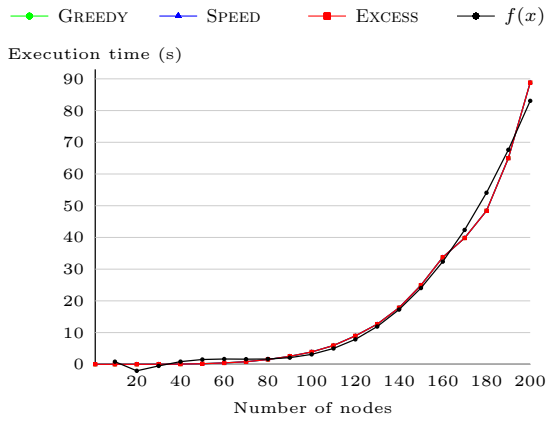


Figure 6: Study of the impact of the number of nodes on the execution time, for random requests between 0 and 100, $\mathcal{P}_{\text{static}} = 50,000$, average on 100 tests.

7. Conclusion

In this paper, we have revisited the well-known replica problem in tree networks under power constraints, in the most flexible scenario where requests of a client can be split between multiple servers. While the problem of minimizing the number of servers has polynomial complexity, we have proved that the problem of minimizing the power consumption is NP-complete, even if the servers are already placed in the tree. We assume that the server speeds can be modified using dynamic voltage and frequency scaling, depending upon the number of requests to be processed, and that a set of discrete speeds is available. Therefore, the core of the difficulty lies in assigning requests to servers in order to optimize the speeds given to each server. Building upon the optimal solution with already placed servers and continuous speeds, we have designed efficient polynomial-time heuristics to solve the general optimization problem (deciding where to place servers and how to assign requests).

In order to assess the performance of the heuristics, we have also provided a mixed integer linear program (MILP) that returns the optimal solution of the problem for small instances (up to 30 nodes in the tree). The heuristics are always quite close to the optimal solution, and the sophisticated versions that readjust request assignments to better fit server speeds prove to be valuable improvements of the basic greedy solution.

For future work, it would be very interesting to prove a competitive ratio for the heuristics that we have designed. However, this is quite a challenging work for arbitrary trees, and one may try to design approximation algorithms only for special tree structures, e.g. binary trees.

Acknowledgements

We would like to thank the reviewers for their comments and suggestions, which greatly improved the final version of this paper. This work was supported in part by the French Research Agency (ANR) through the *Rescue* project. Anne Benoit and Yves Robert are with Institut Universitaire de France.

[1] I. Cidon, S. Kutten, R. Soffer, Optimal allocation of electronic content, Computer Networks 40 (2002) 205–218.

[2] J.-J. Wu, Y.-F. Lin, P. Liu, Optimal replica placement in hierarchical Data Grids with locality assurance, J. Parallel and Distributed Computing 68 (12) (2008) 1517–1538.

[3] A. Benoit, V. Rehn-Sonigo, Y. Robert, Replica placement and access policies in tree networks, IEEE Trans. Parallel and Distributed Systems 19 (12) (2008) 1614–1627.

[4] K. Kalpakis, K. Dasgupta, O. Wolfson, Optimal placement of replicas in trees with read, write, and storage costs, IEEE Trans. Parallel and Distributed Systems 12 (6) (2001) 628–637.

[5] P. Liu, Y.-F. Lin, J.-J. Wu, Optimal placement of replicas in data grid environments with locality assurance, in: Int. Conf. on Parallel and Distr. Syst., 2006.

[6] M. P. Mills, The internet begins with coal, Environment and Climate News.

[7] M. Larabel, Intel EIST SpeedStep.
URL http://www.phoronix.com/

[8] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, D. Takahashi, Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster, in: Proceedings of IPDPS, the IEEE Int. Parallel and Distributed Processing Symposium, 2006. doi:http://doi.ieeecomputersociety.org/10.1109/IPDPS.2006.1639597.

[9] J.-J. Chen, T.-W. Kuo, Multiprocessor energy-efficient scheduling for real-time tasks, in: Proceedings of Int. Conf. on Parallel Proc. (ICPP), IEEE, 2005, pp. 13–20.

[10] A. P. Chandrakasan, A. Sinha, Jouletrack: A web based tool for software energy profiling, in: Design Automation Conference, IEEE, 2001, pp. 220–225.

[11] K. Pruhs, R. van Stee, P. Uthaisombut, Speed scaling of tasks with precedence constraints, Theory of Computing Systems 43 (2008) 67–80.

[12] T. Loukopoulos, I. Ahmad, D. Papadias, An overview of data replication on the Internet, in: Proc. Int. Symp. on Parallel Architectures, Algorithms and Networks ISPAN'02, IEEE Computer Society Press, 2002.

[13] B. Shi, A. Srivastava, Thermal and Power-Aware Task Scheduling and Data Placement for Storage Centric Datacenters, in: Handbook of Energy-Aware and Green Computing, edited by S. Ranka and I. Ahmad, Vol. 1, CRC Press, 2012.

[14] A. Benoit, P. Renaud-Goud, Y. Robert, Power-aware replica placement and update strategies in tree networks, in: Proceedings of IPDPS'2011, the 25th IEEE Int. Parallel and Distributed Processing Symposium, 2011.
URL http://graal.ens-lyon.fr/~abenoit/

[15] R. Xu, D. Mossé, R. Melhem, Minimizing expected energy consumption in real-time systems through dynamic voltage scaling, ACM Trans. Comput. Syst. 25 (4) (2007) 9. doi:http://doi.acm.org/10.1145/1314299.1314300.

[16] M. R. Garey, D. S. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.

[17] G. Aupy, M. Journault, Source code for the simulations, https://github.com/Gaupy/replica.
URL https://github.com/Gaupy/replica

[18] Cplex, ILOG CPLEX: High-performance software for mathematical programming and optimization, http://www.ilog.com/products/cplex/.

[19] J.-J. Chen, Expected energy consumption minimization in DVS systems with discrete frequencies, in: Proc. of SAC'08, Symp. on Applied Computing, 2008, pp. 1720–1725. doi:http://doi.acm.org/10.1145/1363686.1364095.

[20] L. Niu, Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee, in: Proc. of RTCSA, the 16th Int. Conf. on Embedded and Real-Time Computing Systems and App., 2010, pp. 163 –172. doi:10.1109/RTCSA.2010.41.