Power-Aware Replica Placement in Tree Networks with Multiple Servers per Client

Guillaume Aupy[1], Anne Benoit[1], Matthieu Journault[1], and Yves Robert[1,2]

[1] École Normale Supérieure de Lyon, CNRS & INRIA, France
{guillaume.aupy,anne.benoit,yves.robert}@ens-lyon.fr
[2] 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.

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 [8,20,3], with a broad spectrum of applications, such as electronic, ISP, or VOD service delivery (see [12,8,14] and additional references in [20]). 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 classical 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 [16]. 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 [13,11]. 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 [11]. More precisely, a server operated at speed s dissipates s^3 watts [6,5,18]. 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, start from the solution with continuous speeds (including the greedy placement and 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 heuristic 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) exponential cost, but it enables us to assess the absolute performance of the heuristics, at least for small-size problems.

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 [15]. Recently, some work start to tackle energy-related problems. For instance, in [19], 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 [8,14]. Kalpakis et al. [12] 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 [4] 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 [6,5,18].

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. As for 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.

Paper Organization. The rest of the paper is organized as follows. Section 2 is devoted to a precise statement of the framework. Section 3 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 4 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 5 reports experimental 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 Framework

Replica Placement Problem. We consider a distribution tree whose nodes are partitioned into a set of clients $\mathcal{C}$, and a set of N 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 $\mathcal{R} = \{j \in \mathcal{N} | \ \exists i \in \mathcal{C} \ , \ r_{i,j} > 0\}$.

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,

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

and the total power consumption $\mathcal{P}(\mathcal{R})$ of the solution is the sum of the power consumption of all server nodes:

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

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

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 total power consumption.

3 Complexity Results

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

We provide a sketch of the proof here; the detailed proof is very long and technical and can be found in the companion research report [1]. The reduction comes from 2-Partition [10], and the tree consists of a root with n children. There is a server on each node, hence the proof works for both problems. Each child node has a number of requests that is very different from the other children, but depends on the a_i from 2-Partition. The server has therefore the choice between only two speeds among those belonging to the set of possible speeds: either it takes the lower speed and let a_i requests go up in the tree (but the root node can only accommodate $\sum a_i/2$ requests), or it takes the upper speed but the loss in power also is linear in terms of the a_i. The problem then amounts to select which servers run at their lower speed, and it is equivalent to 2-Partition.

Theorem 2. *The following Mixed Integer Linear Program (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 $j \in \mathcal{N}$, $1 \le k \le K$ and $i \in \mathcal{C}$:*

$$\begin{array}{ll}
\textit{Minimize} \sum_{j \in \mathcal{N}} \sum_{1 \le k \le K} x_{j,k}(\mathcal{P}_{static} + s_k^3) \textit{ subject to} & \\
\textit{(i)} \sum_{j \in \mathcal{N}} y_{i,j} = r_i, & i \in \mathcal{C} \\
\textit{(ii)} \sum_{1 \le k \le K} x_{j,k} \le 1, & j \in \mathcal{N} \\
\textit{(iii)} \sum_{i \in \mathcal{C}} y_{i,j} \le \sum_{1 \le k \le K} x_{j,k} s_k, & j \in \mathcal{N}
\end{array} \quad (2)$$

Proof. The constants are the r_i's for $i \in C$, and the s_k's for $1 \le k \le K$, 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 $1 \le k \le K$; $x_{j,k} = 0$ 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 are processed: $\forall i \in \mathcal{C}, \sum_{j \in \mathcal{N}} y_{i,j} = r_i$;
- Each server is assigned at most one speed: $\forall j \in \mathcal{N}, \sum_{1 \le k \le K} x_{j,k} \le 1$; note that a node j is equipped with a server if and only if $\sum_{1 \le k \le K} x_{j,k} = 1$;
- The processing capacity of any server cannot be exceeded: $\forall j \in \mathcal{N}, \sum_{i \in \mathcal{C}} y_{i,j} \le \sum_{1 \le k \le K} x_{j,k} s_k$.

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

4 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.

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, whose proof is long and technical (see the companion research report [1] for full details):

Proposition 1. *Given a fixed set of servers deployed on a tree of size $t = |\mathcal{C}| + |\mathcal{N}|$ 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 $O(t^2)$.*

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. This 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 involves 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.

We provide 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. While simple, Greedy provides a $\left(1 + \frac{\max_i(s_{i+1} - s_i)}{s_{\min}}\right)^3$-approximation for the problem with the placement given by ALG-CONT-PLACED, where $s_{\min}$ is the smallest speed available ($s_{\min} = s_1$). Finally, we point out 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 (see [1] for further details).

The next two heuristics, SPEED and EXCESS, improve the GREEDY heuristic by trying to modify the load of each server, via request re-assignment. The goal is to decrease the speed of some servers. More precisely, in the procedure, which is called EQUILIBRATE and detailed in [1], 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. The capacity of a server is defined as the maximum between its actual speed and the maximum speed of its children, hence we transfer 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.

Recall that K is the number of speeds and $t = |\mathcal{C}| + |\mathcal{N}|$. The complexity of the GREEDY heuristic is $O(t^2)$, 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 $O(Kt^2 \log t)$ [1].

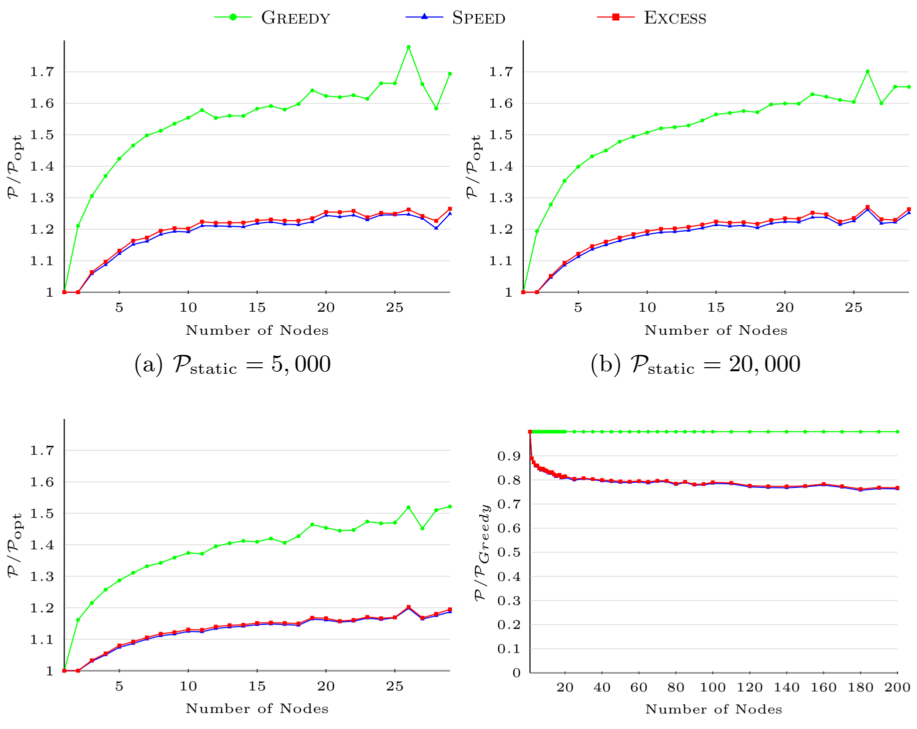
5 Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 4. All the source code, together with scripts to obtain additional results that were omitted due to lack of space, are publicly available [2]. 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 [9].

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. For processor speeds, unless stated otherwise, we use five speeds spaced as those of the Intel Xscale, following [7,17]: we suppose that 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 [7,17], 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 four different sets of simulations to assess the impact of the number of nodes, of static power, of the number of available speeds and of the total load of requests. Note that for the first and the last sets of simulations, additional plots with more values of static power can be found in the companion research report [1].

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, we plot the ratio of the power returned by the heuristics over the power of the optimal solution, with a static power of 5,000, 20,000, and 100,000 respectively. Note that Figure 1d is different from the others and provides results at larger scale: there 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



(a) $\mathcal{P}_{\text{static}} = 5,000$ (b) $\mathcal{P}_{\text{static}} = 20,000$

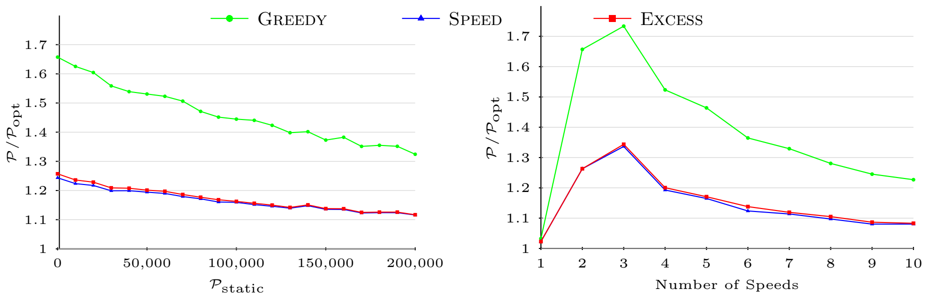
(c) $\mathcal{P}_{\text{static}} = 100,000$ (d) $\mathcal{P}_{\text{static}} = 50,000$, up to 200 nodes

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

of nodes (up to 200 nodes). While one could expect that the performance of the heuristics would decrease for larger trees, it seems that Speed and Excess reach a plateau after $\approx$ 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 solution 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. Note that in Figure 1d, 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.

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 2a, 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 2b, 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



(a) Impact of the static power

(b) Impact of the number of speeds, with $\mathcal{P}_{\text{static}} = 50,000$

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

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 2b, we see that the ratio never goes above 1.8.

Impact of the Total Load of the Tree. Finally, in the last set of simulations, we have studied the impact of the total load of the tree on the power consumption. In Figure 3, 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 5,000 and 100,000 respectively. Overall, 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 (see additional plots in [1]).

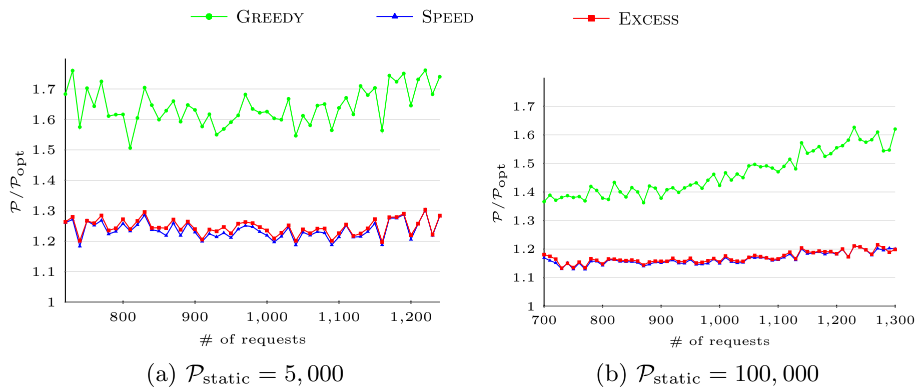


Fig. 3. Study of the impact of the total load, for trees of 20 nodes, for random requests between 0 and 100, average on 100 tests

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 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.

6 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 the request assignment 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. 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.

References

1. Aupy, G., Benoit, A., Journault, M., Robert, Y.: Power-aware replica placement in tree networks with multiple servers per client. Research Report 8474, INRIA (February 2014), `http://graal.ens-lyon.fr/~abenoit/`

2. Aupy, G., Journault, M.: Source code for the simulations, `https://github.com/Gaupy/replica`
3. Benoit, A., Rehn-Sonigo, V., Robert, Y.: Replica placement and access policies in tree networks. IEEE Trans. Parallel and Distributed Systems 19(12), 1614–1627 (2008)
4. Benoit, A., Renaud-Goud, P., Robert, Y.: Power-aware replica placement and update strategies in tree networks. In: Proceedings of the 25th IEEE Int. Parallel and Distributed Processing Symposium, IPDPS 2011 (May 2011)
5. Chandrakasan, A.P., Sinha, A.: Jouletrack: A web based tool for software energy profiling. In: Design Automation Conference. IEEE, pp. 220–225 (2001)
6. Chen, J.J., Kuo, T.W.: Multiprocessor energy-efficient scheduling for real-time tasks. In: Proceedings of Int. Conf. on Parallel Proc (ICPP), pp. 13–20. IEEE (2005)
7. Chen, J.J.: Expected energy consumption minimization in DVS systems with discrete frequencies. In: Proc. of SAC 2008, Symp. on Applied Computing, pp. 1720–1725 (2008)
8. Cidon, I., Kutten, S., Soffer, R.: Optimal allocation of electronic content. Computer Networks 40, 205–218 (2002)
9. Cplex: ILOG CPLEX: High-performance software for mathematical programming and optimization, `http://www.ilog.com/products/cplex/`
10. Garey, M.R., Johnson, D.S.: Computers and Intractability, a Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
11. Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., Takahashi, D.: Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster. In: The IEEE Int. Parallel and Distributed Processing Symposium Proceedings of IPDPS (2006)
12. Kalpakis, K., Dasgupta, K., Wolfson, O.: Optimal placement of replicas in trees with read, write, and storage costs. IEEE Trans. Parallel and Distributed Systems 12(6), 628–637 (2001)
13. Larabel, M.: Intel EIST SpeedStep
14. Liu, P., Lin, Y.F., Wu, J.J.: Optimal placement of replicas in data grid environments with locality assurance. In: Int. Conf. on Parallel and Distr. Syst. (2006)
15. Loukopoulos, T., Ahmad, I., Papadias, D.: An overview of data replication on the Internet. In: Proc. Int. Symp. on Parallel Architectures, Algorithms and Networks ISPAN 2002. IEEE Computer Society Press (2002)
16. Mills, M.P.: The internet begins with coal. Environment and Climate News (1999)
17. Niu, L.: 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., 163 –172 (August 2010)
18. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory of Computing Systems 43, 67–80 (2008)
19. Shi, B., Srivastava, A.: Thermal and Power-Aware Task Scheduling and Data Placement for Storage Centric Datacenters. In: Ranka, S., Ahmad, I. (eds.) Handbook of Energy-Aware and Green Computing, vol. 1, CRC Press (2012)
20. Wu, J.J., Lin, Y.F., Liu, P.: Optimal replica placement in hierarchical Data Grids with locality assurance. J. Parallel and Distributed Computing 68(12), 1517–1538 (2008)