Impact of QoS on Replica Placement in Tree Networks

Anne Benoit, Veronika Rehn, and Yves Robert

Laboratoire LIP, ENS Lyon, France. UMR CNRS-INRIA-UCBL 5668
{Anne.Benoit|Veronika.Rehn|Yves.Robert}@ens-lyon.fr

Abstract. This paper discusses and compares several policies to place replicas in tree networks, subject to server capacity and QoS constraints. The client requests are known beforehand, while the number and location of the servers are to be determined. We study three strategies. The first two strategies assign each client to a unique server while the third allows requests of a client to be processed by multiple servers. The main contribution of this paper is to assess the impact of QoS constraints on the total replication cost. In this paper, we establish the NP-completeness of the problem on homogeneous networks when the requests of a given client can be processed by multiple servers. We provide several efficient polynomial heuristic algorithms for NP-complete instances of the problem.

Keywords: Replica placement, QoS constraints, access policies, heterogeneous platforms, complexity, placement heuristics.

1 Introduction

This paper deals with the problem of replica placement in tree networks with Quality of Service (QoS) guarantees. Informally, there are clients issuing several requests per time-unit, to be satisfied by servers with a given QoS. The clients are known (both their position in the tree and their number of requests), while the number and location of the servers are to be determined. A client is a leaf node of the tree, and its requests can be served by one or several internal nodes. Initially, there are no replicas; when a node is equipped with a replica, it can process a number of requests, up to its capacity limit (number of requests served by time-unit). 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. Every client has some QoS constraints: its requests must be served within a limited time, and thus the servers handling these requests must not be too far from the client.

The rule of the game is to assign replicas to nodes so that some optimization function is minimized and QoS constraints are respected. Typically, this optimization function is the total utilization cost of the servers. In this paper we study this optimization problem, called REPLICA PLACEMENT, and we restrict

the QoS in terms of number of hops. This means for instance that the requests of a client who has a QoS range of $qos = 5$ must be treated by one of the first five internal nodes on the path from the client up to the tree root.

We point out that the distribution tree (clients and nodes) is fixed in our approach. This key assumption is quite natural for a broad spectrum of applications, such as electronic, ISP, or VOD service delivery. The root server has the original copy of the database but cannot serve all clients directly, so a distribution tree is deployed to provide a hierarchical and distributed access to replicas of the original data. On the contrary, in other, more decentralized, applications (e.g. allocating Web mirrors in distributed networks), a two-step approach is used: first determine a “good” distribution tree in an arbitrary interconnection graph, and then determine a “good” placement of replicas among the tree nodes. Both steps are interdependent, and the problem is much more complex, due to the combinatorial solution space (the number of candidate distribution trees may well be exponential).

Many authors deal with the REPLICA PLACEMENT optimization problem. Most of the papers do not deal with QoS but instead consider average system performance such as total communication cost or total accessing cost. Please refer to [2] for a detailed description of related work with no QoS constraints.

Cidon et al [4] studied an instance of REPLICA PLACEMENT with multiple objects, where all requests of a client are served by the closest replica (*Closest* policy). In this work, the objective function integrates a communication cost, which can be seen as a substitute for QoS. Thus, they minimize the average communication cost for all the clients rather than ensuring a given QoS for each client. They target fully homogeneous platforms since there are no server capacity constraints in their approach. A similar instance of the problem has been studied by Liu et al [7], adding a QoS in terms of a range limit, and whose objective is to minimize the number of replicas. In this latter approach, the servers are homogeneous, and their capacity is bounded. Both [4,7] use a dynamic programming algorithm to find the optimal solution.

Some of the first authors to introduce actual QoS constraints in the problem were Tang and Xu [9]. In their approach, the QoS corresponds to the latency requirements of each client. Different access policies are considered. First, a replica-aware policy in a general graph with heterogeneous nodes is proven to be NP-complete. When the clients do not know where the replicas are (replica-blind policy), the graph is simplified to a tree (fixed routing scheme) with the *Closest* policy, and in this case again it is possible to find an optimal dynamic programming algorithm. In [10], Wang et al deal with the QoS aware replica placement problem on grid systems. In their general graph model, QoS is a parameter of communication cost. Their research includes heterogeneous nodes and communication links. A heuristic algorithm is proposed and compared to the results of Tang and Xu [9].

Another approach, this time for dynamic content distribution systems, is proposed by Chen et al [3]. They present a replica placement protocol to build a dissemination tree matching QoS and server capacity constraints. Their work

focuses on Web content distribution built on top of peer-to-peer location services: QoS is defined by a latency within which the client has to be served, whereas server capacity is bounded by a fan-out-degree of direct children. Two placement algorithms (a native and a smart one) are proposed to build the dissemination tree over the physical structure.

In [2] we introduced two new access policies besides the *Closest* policy. In the first one, the restriction that all requests from a given client are processed by the same replica is kept, but client requests are allowed to “traverse” servers in order to be processed by other replicas located higher in the path (closer to the root). This approach is called the *Upwards* policy. In the second approach, access constraints are further relaxed and the processing of a given client’s requests can be split among several servers located in the tree path from the client to the root. This policy with multiple servers is called *Multiple*.

In this paper we study the impact of QoS constraints on these three policies. On the theoretical side we prove the NP-completeness of *Multiple*/Homogeneous instance with QoS constraints, while the same problem was shown to be polynomial without QoS [2]. This result shows the additional combinatorial difficulties which we face when enforcing QoS constraints. On the practical side, we propose several heuristics for all policies. We compare them through simulations conducted for problem instances with different ranges of QoS constraints. We are also able to assess the absolute performance of the heuristics, by comparing them to the optimal solution of the problem provided by a formulation of the REPLICAS PLACEMENT problem in terms of a mixed integer linear program. The solution of this program allows us to build an optimal solution [1] for reasonably large problem instances.

2 Framework and Access Policies

We consider a distribution tree $\mathcal{T}$ 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. A client $i \in \mathcal{C}$ is making r_i requests per time unit to a database, with a QoS qos_i : the database must be placed not further than qos_i hops on the path from the client to the root.

A node $j \in \mathcal{N}$ may or may not have been provided with a replica of the database. A node j equipped with a replica (*i.e.* j is a server) can process up to W_j requests per time unit from clients in its subtree. In other words, there is a unique path from a client i to the root of the tree, and each node in this path is eligible to process some or all the requests issued by i when provided with a replica. We denote by $R \subseteq \mathcal{N}$ the set of replicas, and $\text{Servers}(i) \subseteq R$ is the set of nodes which are processing requests from client i . The number of requests from client i satisfied by server s is $r_{i,s}$, and the number of hops between i and $j \in \mathcal{N}$ is denoted by $d(i, j)$. Two constraints must be satisfied:

- Server capacity: $\forall s \in R, \quad \sum_{i \in \mathcal{C} | s \in \text{Servers}(i)} r_{i,s} \leq W_s$
- QoS constraint: $\forall i \in \mathcal{C}, \forall s \in \text{Servers}(i), \quad d(i, s) \leq \text{qos}_i$

The objective function for the REPLICA PLACEMENT problem is defined as: $\text{Min} \sum_{s \in R} W_s$. When the servers are homogeneous, *i.e.* $\forall s \in \mathcal{N}, W_s = W$, the optimization problem reduces to finding a minimal number of replicas. This problem is called REPLICA COUNTING.

We consider three access policies in this paper. The first two are single server strategies, *i.e.* each client is assigned a single server responsible for processing all its requests. The *Closest* policy is the most restricted one: the server for client i is enforced to be the first server that can be found on the path from i upwards to the root. Relaxing this constraint leads to the *Upwards* policy. Clients are still assigned to a single server, but their requests are allowed to traverse one or several servers on the way up to the root, in order to be processed by another server closer to the root. The third policy is a multiple server strategy and hence a further relaxation: a client i may be assigned a set of several servers. Each server $s \in \text{Servers}(i)$ will handle a fraction $r_{i,s}$ of requests. Of course $\sum_{s \in \text{Servers}(i)} r_{i,s} = r_i$. This policy is referred to as the *Multiple* policy.

3 Complexity Results

Table 1 gives an overview of complexity results of the different instances of the REPLICA COUNTING problem (homogeneous servers). Liu et al [7] provided a polynomial algorithm for the *Closest* policy with QoS constraints. In [2] we proved the NP-completeness of the *Upwards* policy without QoS. This was a surprising result, to be contrasted with the fact that the *Multiple* policy is polynomial under the same conditions [2].

Table 1. Complexity results for the different instances of REPLICA COUNTING

	Homogeneous	Homogeneous/QoS
<i>Closest</i>	polynomial [4,7]	polynomial [7]
<i>Upwards</i>	NP-complete [2]	NP-complete [2]
<i>Multiple</i>	polynomial [2]	NP-complete (this paper)

An important contribution of this paper is the NP-completeness of the *Multiple* policy with QoS constraints. As stated above, the same problem was polynomial without QoS, which gives a clear insight on the additional complexity introduced by QoS constraints. The proof uses a reduction to 2-PARTITION-EQUAL [5]. Due to a lack of space we refer to the extended version of this paper [1] for the complete proof.

Theorem 1. *The instance of the REPLICA COUNTING problem with QoS constraints and the Multiple strategy is NP-complete.*

Finally, we point out that all three instances of the REPLICA PLACEMENT problem (heterogeneous servers with the *Closest*, *Upwards* and *Multiple* policies) are already NP-complete without QoS constraints [2].

4 Heuristics for the Replica Placement Problem

In this section several heuristics for the *Closest*, *Upwards* and *Multiple* policies are presented. As already pointed out, the quality of service is the number of hops that requests of a client are allowed to traverse until they have to reach their server. The code and some more heuristics can be found on the web [8]. All heuristics described below have polynomial, and even worst-case quadratic, complexity $O(s^2)$, where $s = |\mathcal{C}| + |\mathcal{N}|$ is the problem size.

In the following, we denote by inreqQoS_j the amount of requests that reach an inner node j within their QoS constraints, and by inreq_j the total amount of requests that reach j (including requests whose QoS constraints are violated).

Closest Big Subtree First - CBS. Here we traverse the tree in top-down manner. We place a replica on an inner node j if $\text{inreqQoS}_j \leq W_j$. When the condition holds, we do not process any other subtree of j . If this condition does not hold, we process the subtrees of j in non-increasing order of inreq_j . Once no further replica can be added, we repeat the procedure. We stop when no new replica is added during a pass.

Upwards Small QoS Started Servers First - USQoS. Clients are sorted by non-decreasing order of qos_i (and non-increasing order of r_i in case of tie). For each client i in the list we search for an appropriate server: we take the next server on the way up to the root (i.e. an inner node that is already equipped with a replica) which has enough remaining capacity to treat all the client's requests. Of course the QoS-constraints of the client have to be respected. If there is no server, we take the first inner node j that satisfies $W_j \geq r_i$ within the QoS-range and we place a replica in j . If we still find no appropriate node, this heuristic has no feasible solution.

Upwards Minimal Distance - UMD. This heuristic requires two steps. In the first step, so-called indispensable servers are chosen, i.e. inner nodes which have a client that must be treated by this very node. At the beginning, all servers that have a child client with $\text{qos} = 1$ will be chosen. This step guarantees that in each loop of the algorithm, we do not forget any client. The criterion for indispensable servers is the following: for each client check the number of nodes eligible as servers; if there is only one, this node is indispensable and chosen. The second step of UMD chooses the inner node with minimal $(W_j - \text{inreqQoS}_j)$ -value as server (if $\text{inreqQoS}_j > 0$). Note that this value can be negative. Then clients are associated to this server in order of distance, i.e. clients that are close to the server are chosen first, until the server capacity W_j is reached or no further client can be found.

Multiple Small QoS Close Servers First - MSQoSC. The main idea of this heuristic is the same as for USQoS, but with two differences. Searching for an appropriate server, we take the first inner node on the way up to the root which has some remaining capacity. Note that this makes the difference between *close* and *started* servers. If this capacity W_i is not sufficient (client c has more requests, $W_i < r_c$), we choose other inner nodes going upwards to the root until all requests of the client can be processed (this is possible owing to

the multiple-server relaxation). If we cannot find enough inner nodes for a client, this heuristic will not return a feasible solution.

Multiple Small QoS Minimal Requests - MSQoS_M. In this heuristic clients are treated in non-decreasing order of qos_i , and the appropriate servers j are chosen by minimal $(W_j - \text{inreqQoS}_j)$ -value until all requests of clients can be processed.

Multiple Minimal Requests - MMR. This heuristic is the counterpart of UMD for the *Multiple* policy and requires two steps. Servers are added in the “indispensable” step, either when they are the only possible server for a client, or when the total capacity of all possible inner nodes for a client i is exactly r_i . The server chosen in the second step is also the inner node with minimal $(W_j - \text{inreqQoS}_j)$ -value, but this time clients are associated in non-decreasing order of $\min(qos_i, d(i, r))$, where $d(i, r)$ is the number of hops between i and the root of the tree. Note that the last client that is associated to a server, might not be processed entirely by this server.

Mixed Best - MB. This heuristic unifies all previous ones, including those presented in [1]. For each tree, we select the best cost returned by the other heuristics. Since each solution for *Closest* is also a solution for *Upwards*, which in turn is a valid solution for *Multiple*, this heuristic provides a solution for the *Multiple* policy.

5 Experimental Plan

In this section we evaluate the performance of our heuristics on tree platforms with varying parameters. Through these experiments we want to assess the different access policies, and the impact of QoS constraints on the performance of the heuristics. We obtain an optimal solution for each tree platform with the help of a mixed integer linear program, see [1] for further details. We can compute the latter optimal solution for problem sizes up to 400 nodes and clients, using GLPK [6].

An important parameter in our tree networks is the load, i.e. the total number of requests compared to the total processing power: $\lambda = \frac{\sum_{i \in \mathcal{C}} r_i}{\sum_{j \in \mathcal{N}} W_j}$, where $\mathcal{C}$ is the set of clients in the tree and $\mathcal{N}$ the set of inner nodes. We tested our heuristics for $\lambda = 0.1, 0.2, \dots, 0.9$, each on 30 randomly generated trees of two heights: in a first series, trees have a height between 4 and 7 (small trees). In the second series, tree heights vary between 16 and 21 (big trees). All trees have s nodes, where $15 \leq s \leq 400$. To assess the impact of QoS on the performance, we study the behavior (i) when QoS constraints are very tight ($qos \in \{1, 2\}$); (ii) when QoS constraints are more relaxed (the average value is set to half of the tree height height); and (iii) without any QoS constraint ($qos = \text{height} + 1$).

We have computed the number of solutions for each λ and each heuristic. The number of solutions obtained by the linear program indicates which problems are solvable. Of course we cannot expect a result with our heuristics for intractable problems. To assess the performance of our heuristics, we have studied the relative performance of each heuristic compared to the optimal solution. For each λ ,

Fig. 1. Relative performance of small trees with (a) tight, (b) medium, (c) no QoS constraints. (d) Big trees with medium QoS constraints.

the cost is computed on those trees for which the linear program has a solution. Let T_λ be the subset of trees with a LP solution. Then, the relative performance for the heuristic h is obtained by $\frac{1}{|T_\lambda|} \sum_{t \in T_\lambda} \frac{\text{cost}_{LP}(t)}{\text{cost}_h(t)}$, where $\text{cost}_{LP}(t)$ is the optimal solution cost returned by the linear program on tree t , and $\text{cost}_h(t)$ is the cost involved by the solution proposed by heuristic h . In order to be fair versus heuristics that have a higher success rate, we set $\text{cost}_h(t) = +\infty$, if the heuristic did not find any solution.

Figure 1 gives an overview of our performance tests (see [1] for the complete set of results). The comparison between Fig. 1a and 1c shows the impact of QoS on the performance. The impact of the tree sizes can be seen by comparing Fig. 1b and 1d. Globally, all the results show that QoS constraints do not modify the relative performance of the three policies: with or without QoS, *Multiple* is better than *Upwards*, which in turn is better than *Closest*, and their difference in performance is not sensitive to QoS tightness or to tree sizes. This is an enjoyable result, that could not be predicted a priori. The MB heuristic returns very good results, being relatively close to the optimal in most cases. The best heuristic to use depends on the tightness of QoS constraints. Thus, for *Multiple*, MSQoSM is the best choice for tight QoS constraints and small λ (Fig. 1a). When QoS is less constrained, MMR is the best for λ up to 0.4. For big λ , MSQoSC is to prefer, since it never performs poorly in this case. Concerning the *Upwards* policy, USQoSS behaves the best for tight QoS, in the other cases UMD achieves better results. We kept only our best *Closest* heuristic on these curves, which outperforms the others [1].

6 Conclusion

In this paper we have dealt with the REPLICA PLACEMENT optimization problem with QoS constraints. We have proven the NP-completeness of *Multiple* with QoS constraints on homogeneous platforms, and we have proposed a set of efficient heuristics for the *Closest*, *Upwards* and *Multiple* access policies. To evaluate the absolute performance of our algorithms, we have compared the experimental results to the optimal solution of an integer linear program, and these results turned out quite satisfactory. In our experiments we have assessed the impact of QoS constraints on the different policies, and we have discussed which heuristic performed best depending upon problem instances, platform parameters and QoS tightness. We have also showed the impact of platform size on the performances. Although we studied the problem with a restricted QoS model, but we expect experimental results to be similar for more general QoS constraints.

As for future work, bandwidth and communication costs could be included in the experimental plan. Also the structure of the tree networks has to be studied more precisely. In this paper we have investigated different tree heights, but it would be interesting to study the impact of the average degree of the nodes onto the performance. In a longer term, the extension of the REPLICA PLACEMENT optimization problem to various object types should be considered, which would call for the design and evaluation of new efficient heuristics.

References

1. Benoit, A., Rehn, V., Robert, Y.: Impact of QoS on Replica Placement in Tree Networks. Research Report 2006-48, LIP, ENS Lyon, France, available at graal.ens-lyon.fr/~yrobert/ (2006).
2. Benoit, A., Rehn, V., Robert, Y.: Strategies for Replica Placement in Tree Networks. Research Report 2006-30, LIP, ENS Lyon, France, available at graal.ens-lyon.fr/~yrobert/ (2006)
3. Chen, Y., Katz, R.H., Kubiatowicz, J.D.: Dynamic Replica Placement for Scalable Content Delivery. In: Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA (2002) 306–318
4. Cidon, I., Kutten, S., Soffer, R.: Optimal allocation of electronic content. *Computer Networks* **40** (2002) 205–218
5. Garey, M.R., Johnson, D.S.: *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company (1979)
6. GLPK: GNU Linear Programming Kit. <http://www.gnu.org/software/glpk/>
7. Liu, P., Lin, Y.F., Wu, J.J.: Optimal placement of replicas in data grid environments with locality assurance. In: International Conference on Parallel and Distributed Systems (ICPADS). IEEE Computer Society Press, (2006)
8. Source Code for the Heuristics. <http://graal.ens-lyon.fr/~vrehn/code/replicaQoS/>
9. Tang, X., Xu, J.: QoS-Aware Replica Placement for Content Distribution. *IEEE Trans. Parallel Distributed Systems* **16(10)** (2005) 921–932
10. Wang, H., Liu, P., Wu, J.J.: A QoS-aware Heuristic Algorithm for Replica Placement. In: Proceedings of the 7th International Conference on Grid Computing, GRID2006. IEEE Computer Society, (2006) 96–103