Distributed Dynamic Replica Placement and Request Redirection in Content Delivery Networks

Francesco Lo Presti
Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata"
E-mail: lopresti@info.uniroma2.it

Chiara Petrioli and Claudio Vicari
Dipartimento di Informatica
Università di Roma "La Sapienza"
E-mail: {petrioli,vicari}@di.uniroma1.it

***Abstract*—The Content Delivery Networks (CDN) paradigm is based on the idea to transparently move third-party content closer to the users. More specifically, content is replicated on CDN servers which are located close to the final users, and user requests are redirected to the "best" replica (e.g., the closest) in a transparent way, so that users perceive a better content access service.**

In this paper we address user requests redirection and replica placement in CDNs. Differently from previous solutions our scheme considers the two problems jointly and relies on distributed and localized schemes that can be implemented with little complexity and overhead, thus providing a new overall solution that effectively trades-off among the number of replicas, their utilization (i.e., how many users requests they serve), the distance from the best replica and the number of replica adds and removals.

An OPNET based thorough performance evaluation has allowed us to assess the effectiveness of the proposed solution. By properly tuning the distributed heuristics parameters the CDN provider can have a strict control on the CDN network operations so that the desired trade-off between all the relevant performance metrics is achieved.

I. Introduction

Content Delivery Networks (CDNs) are one of the answers to the challenges posed by the remarkable commercial success of the Internet. Replicating third-party content on servers closer to the final users, and redirecting transparently the users' requests to the "best replica" (e.g., the closest replica in terms of distance, latency, etc.) CDN providers are able to offer improved content access service.

Solutions for CDN thus require to address a number of technical problems, which include: deciding the kind of content that should be hosted (if any) at a given CDN server (replica placement), selecting the best replica for a given user, and redirecting the users requests to the replica.

Building on previous work on dynamic replica placement [1], in this paper we present simple, distributed, scalable algorithms for dynamic replica placement and requests redirection. Our solutions are dynamic in the sense that: 1) replicas are added and removed from CDN servers according to the dynamically changing user request traffic; and 2) requests redirection is adjusted as to achieve load balancing among the different replicas. Our contribution here improves over previous results in that it considers the two problems jointly and relies on distributed and localized schemes that can be implemented with little complexity and overhead, thus providing a new overall solution that effectively trades-off among the number of replicas, their utilization (i.e., how many users requests they serve), the distance from the best replica and the number of replica adds and removals.

The replica placement problem has been widely investigated in the literature. For the static case, simple efficient greedy solutions have been proposed in [2] and [3]. In particular in [2] Qiu et al. have formulated the static replica placement problem as a minimum K median problem. A simple greedy heuristic is shown to have performance within 50% of the optimal strategy. In [3] and [4] Jamin et al. and Radoslavov et al. propose fan-out based heuristics in which replicas are placed at the nodes with the highest fan-out irrespective of the actual cost function. A major limit of all the previously proposed static solutions is that they neglect to consider the natural dynamics in the user requests traffic. In the performance evaluation section we will show that adopting static solutions in a realistic setting where users traffic changes over time either results in poor performance or into high maintenance costs. The former occurs in case replica placement is computed only once (or seldomly recomputed) and the same replica configuration is used for long times independently of current user requests. The latter reflects the case in which static algorithms are executed frequently to try to follow users dynamics, demanding for frequent replicas add/removals. Frequent re-execution of the static replica placement algorithms such as [2], [3], [4] also results into high overhead to periodically gather the information needed to make centralized decisions on the new replica placement.

A few recent works have started addressing the dynamic case [5], [6], [1], [7], proposing solutions which explicitly consider the current replica placement and the reconfiguration costs when deciding which replicas to add or remove to reflect the current and expected future users needs.

The solution for dynamic replica placement proposed in [6] is tightly coupled with the Tapestry architecture it has been designed for. Also, the proposed solution does not explicitly account for neither the costs of reconfiguration nor for possible servers storage limits. In RaDar [5] a threshold based heuristic is proposed to replicate, migrate and delete replicas in response to system dynamics, and such heuristic is combined with a scheme for user requests redirection. However, the proposed

Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'07)
978-1-4244-1854-1/08/$25.00 ©2008 IEEE

scheme does not account for limits on the servers storage and for users QoS constraints (e.g., maximum latency, distance from the serving replica). In [1] we have overcome these limits by introducing a general problem formulation which accounts both for limits on servers storage and load, and for users QoS constraints. The paper introduces both a centralized and a threshold-based distributed heuristic for dynamic replica placement, presenting a preliminary performance evaluation of the proposed solutions. Recently, a similar approach has been applied to dynamically placing, removing and migrating copies of a given content in a hierarchical proxy-caches network (see [8]). The idea is to attract (push back) copies of a content from (to) a given zone in the network depending on the number of requests for that content originated in the zone, overall dynamically optimizing the contents stored in the caches. The optimization aims at minimizing the total distance between the content replicas and the locations where requests for the content are originated.

The solution presented in this paper improves over what has been previously proposed as it jointly addresses user requests redirection and replica placement and it introduces a way to enforce a strict control on the replicas level of utilization. Not only we minimize the costs for replicas placement and maintenance, not only we try to keep as low as possible the number of replicas adds and removals while satisfying all user requests but we do it distributely, load balancing the traffic among replicas and cloning (removing) replicas whenever their level of utilization is above (below) a desirable level of utilization. This provides a powerful tool to the CDN provider: setting the bounds of the replicas' utilization interval as well as the other parameters of our heuristics the CDN provider can have a strict control on how the CDN network will operate and can achieve different trade-offs between all the relevant performance metrics.

The paper is organized as follows. In section II we formulate the problem and introduce the notation we use throughout the paper. Sections III and IV detail the operations of the distributed heuristics we use for sake of replica placement and user requests redirection, respectively. Section V presents the outcomes of a thorough performance evaluation aimed at assessing the performance of the proposed solutions in realistic settings, under varying network topologies, traffic loads and parameter settings. Finally, section VI concludes the paper.

II. Problem formulation

We consider a CDN network hosting a set C of contents. Users access the CDN network through a set V_A of access nodes. The number of users accesses is expressed in units of aggregate requests from that access site, for each type of content. Replicas of the C contents can be stored in one or more sites among a set V_R of CDN servers sites. Each site $j \in V_R$ can host up to V^R_{max} replicas (storage limit), each satisfactorly serving requests as long as the replica load is below a threshold $U_{\max}$ (load threshold). A weight is associated to the route from a user (access node) i to a replica j. The weight $d(i,j)$ indicates the user perceived quality of accessing that replica. A user is said to be satisfied when the weight of the route to the best replica is below a given threshold $d_{\max}$. Each access node i has therefore associated a set $\rho(i)$ which include all the server sites able to satisfy users requests generated at i.

We denote by $x_{i,c}$ the volume of user requests originated at node $i \in V_A$ for content $c \in C$ and by $\alpha_{ij,c}$, $\sum_{j\in\rho(i)} \alpha_{ij,c} = 1$, the fraction of requests for content c, originated at node i, which are redirected to node j. We denote by $r_{j,c}$ the amount of replicas (resources) allocated to content c at node $j \in V_R$. The replica placement and requests redirection goal is to identify a strategy which dynamically allocates and deallocates replicas and redirect requests in response to users demand variations so that the overall cost (overall number of replicas) is minimized while satisfying the users requests and meeting the constraints on the replica service capacity and site resources.

III. Distributed heuristic

In this section we describe a distributed scheme to allocate and deallocate replicas, so that the user requests are satisfied while minimizing the CDN costs in a dynamic scenario. This scheme always accounts for the current replica placement, adding replicas or changing replica location only when needed. Each site $j \in V_R$ autonomously decides on whether some of the replicas it stores should be cloned or removed.

This decision is based on local information: the number and content of the replicas stored at j, the load of such replicas and the user requests they are currently serving. We also assume that each site j stores information about the number and the content of the replicas hosted in its local neighborhood. More specifically, site j knows the set $\alpha(j) \subseteq V_A$ of the access nodes which are distant at most d_{max} from it, and the set $\rho(j) \subseteq V_R$ of the server sites in V_R that are distant at most d_{max} from any of the nodes in $\alpha(j)$. The first set includes all those access sites which can generate requests that j can satisfy. The set $\rho(j)$ is the set of server sites that can cover for j.

Based on this information, site j is able to decide whether to clone or delete a replica, and in case of cloning, where the clone should be hosted. The way cloning and replica removal work is the following. Replicas are classified as *overloaded*, *severely underloaded* and *under the target utilization level* depending on the current load. They belong to the first category whenever their load exceeds a threshold U_{max}. User requests served by overloaded replicas might suffer severe performance degradation. To avoid persistent replica overloading each site will clone replicas which are above U_{max}. On the other hand, if the replica load is below a threshold U_{low} (i.e., the replica is underloaded), the provider pays for a replica which offers little contribution to the system operations. The distributed algorithm we propose tries to delete underloaded replicas. The underloaded replica first tries to direct away all requests it is currently serving, by providing the access nodes with a false feedback making them believe their requests could be better served by a different replica. If all the currently served user

requests can be redirected to other replicas (the replica has no requests to serve) then the replica is safely deleted.

A less critical (but common) situation is that in which the replica serves an adequate number of user requests ($> U_{low}$) even if its current load is below what desirable to justify the costs for replica maintenance (i.e. it is below a threshold U_{mid}). This is the case of a replica under the target utilization level. To cope with this case we designed a distributed probabilistic mechanism which tries to redirect the user requests and delete replicas over time so that the load of current replicas is kept in the range ($U_{mid} < l < U_{max}$). This mechanism is the core of our solution since it allows us to specify the desirable utilization range for the replicas. Replicas will be dynamically allocated and deallocated in order to satisfy all user requests but also in order to ensure that each replica the CDN provider pays for is properly utilized.

In the following we detail the distributed algorithms for cloning (algorithm 1) and removing replicas, and the U_{mid} based distributed probabilistic mechanism.

A. Cloning of a replica

The function to clone a replica of content $c \in C$ (algorithm 1 below) is called by a server site j whenever the load of one of its replicas of content c exceeds U_{max}. The function outputs the server sites(s) $best_{vr}$ where the cloned replica(s) should be added. The detailed operations of the function *add_replica* are reported below.

Algorithm 1 Function add_replica(j, c)

Require: $j \in V_R$

1: $l_{j,c} = \sum_{i \in V_A} \alpha_{ij,c} \cdot x_{i,c}$
2: **while** $\frac{l_{j,c}}{r_{j,c}} - U_{max} > 0$ **do**
3: $best\_served = 0$
4: $best\_distance = \infty$
5: $best\_vr = undefined$
6: **for all** $j' \in \rho(j)\ s.t.\ r_{j,c} < V_R^{max}$ **do**
7: $l'_{j',c} = \sum_{i \in \alpha(j')} \alpha_{ij,c} \cdot x_{i,c}$
8: $total\_distance = \sum_{i \in \alpha(j'))} \alpha_{ij,c} \cdot x_{i,c} \cdot d_{i,j'}$
9: **if** $(l'_{j',c} < best\_served) \vee$
10: $(l'_{j',c} = best\_served \wedge$
11: $total\_distance < best\_distance)$ **then**
12: $best\_distance = total\_distance$
13: $best\_served = l'_{j',c}$
14: $best\_vr = j'$
15: **end if**
16: **end for**
17: **if** $best\_vr = undefined$ **then**
18: exit
19: **end if**
20: ask best_vr to add a replica
21: compute $l''_{best_{vr},c} = min(\sum_{i \in \alpha(best_{vr})} \alpha_{ij,c} \cdot x_{i,c}, 1)$
22: $l_{j,c} = l_{j,c} - l''_{best_{vr},c}$
23: remove from the set of requests those that can be offloaded
24: **end while**

When the function is invoked node j first computes the number $l_{j,c}$ of requests for content c it currently serves (line 1). If the average load of the replicas for content c hosted at node j is above the threshold U_{max} (line 2) then a new replica of content c will be added to the network with the aim to offload overloaded replicas (lines 3–16). The new replica location $best_{vr} = j'$ is chosen based on $l'_{j',c}$, which is defined as the number of user requests currently served by a replica hosted at node j which could be offloaded to a new replica added at j'. The higher the value of $l'_{j',c}$ the more suited j' is to host the new replica. Ties are broken by selecting, among those sites maximizing $l'_{j',c}$ and having space for an additional replica, the site which maximizes the satisfaction of the offloaded users requests i.e., the site which minimizes the total distance $\sum_{i \in \alpha(j')} \alpha_{ij,c} \cdot x_{i,c} \cdot d_{i,j'}$ (lines 8–14).

If a suitable site $best_{vr}$ for hosting the clone is found node j sends a message to it asking to add a replica of content c (line 20). Node j also computes the maximum amount $l''_{best_{vr},c} = min(\sum_{i \in \alpha(best_{vr})} \alpha_{ij,c} \cdot x_{i,c}, 1)$ of requests it is currently serving that can be offloaded to $best_{vr}$ without overloading the new replica (lines 20–23). It then checks whether the requests j is serving and that cannot be offloaded to $best_{vr}$ are still too many, i.e., if they still overload node j replicas (line 23). If this is the case an additional replica is added to the network and the procedure is re-executed on the requests that cannot be offloaded.

It might happen that the whole system (or a portion of it) is overloaded so that no site at which a replica can be added is identified (lines 17–18). In this case the algorithm has no way to improve the situation and the procedure is exited.

B. Replica removal

A replica can be removed if (and only if) it serves no requests. In order to favor replica removal minimizing the number of replicas, every server site j assumes that its first $r_{j,c} - 1$ replicas serve U_{max} requests, while only $l_{j,c} - ((r_{j,c} - 1) * U_{max})$ requests are served by the last replica. The last replica can be removed if the other replicas hosted at the site can cover for it.

To gain a finer control over the mechanism and avoid situations in which a temporary traffic decrease triggers the removal of a replica which will soon be needed again, we base replica removal on the exponential average of the replica past and current load. A replica is removed only when it has not been serving requests for a time long enough to bring the exponential average down to zero.

C. Handling replicas under the target utilization level

When replicas are below the target utilization level, but still over U_{low}, a distributed probabilistic mechanism is employed in order to redistribute the load. To make an example of why replicas can fall below the target utilization level, consider the case in which a high volume of user traffic requests forces the CDN provider to place many replicas to be able to satisfy all user requests. If the user requests volume decreases and user requests are load balanced among the existing replicas

(via the scheme shown in the next section) then the utilization of the different existing replicas will uniformly decrease. Until the traffic decrease is so significant that replicas starts falling below the U_{low} threshold, no replica removal would be performed according to the replica removal algorithm. However, in such a situation it might be possible to remove a significant number of replicas while still being able to satisfy all user requests. To achieve this goal in our scheme a replica whose load is below the target utilization U_{mid} will try to direct requests away (without affecting underloaded and overloaded replicas). Let j be a site at which a replica below the target utilization level is hosted. When providing feedbacks to the access sites i on how effectively requests generated at i can be served by the replica hosted at j, node j will first toss a coin and with a probability $p_{l_{j,c}}$ will decide to 'cheat' advertising a bad service (still better than what is advertised by underloaded or overloaded replicas). The higher the load of a replica the lower the probability it will cheat. The cheating replicas will then be partially offloaded provided that there are other replicas that can cover for them whose load is between U_{low} and U_{max}. As their load is decreased they will have even higher probabilities of directing further requests away. Eventually such replicas will redirect away all the requests they're serving and will be removed (if possible).

D. The bootstrap case

The last case to be considered is the following. It might happen that a user requests access to a content c from an access site i which does not have any replica of content c within distance d_{max} (this is often the case when the replica allocation process starts). In this case the request is directed to the origin server, that clones a copy of its content to a replica site j that is distant at most d_{max} from i. Among the possible sites, the origin server selects the site j that can satisfy the requests originated by the largest number of sites of V_A. The selected site is clearly highly likely to be able to satisfy the largest number of requests in the near future. Ties are broken by selecting a node that minimizes the overall average distance from the requests.

IV. Redirection algorithm

We assumed that the system is capable of redirecting the requests obtaining a best effort load balancing of the load among replicas. (A perfect load balancing may be impossible due to the distance constraint.)

The redirection system exploits and accounts for feedback it receives from the replicas on the current replicas loads (and associated expected latencies to access the content). Based on this feedback users requests are directed away from a replica in response to threshold events (if the replica load exceeds U_{max} or falls below U_{low}). As an example, an underloaded replica informs the redirection system which then tries to offload requests to some other replicas (if possible).

Redirection. User requests are redirected to available replicas. For a site $j \in V_R$, let $l_{j,c}$ denote the aggregate demand of content c served by j. The users requests re-directed to j can be fully served if $l_{j,c} \leq r_{j,c}$, i.e., if the aggregate load for that content does not exceed 100%; otherwise, users requests for content c that were redirected to server site j would suffer some level of service degradation.

Load Balancing The baseline approach consists in achieving load balancing among replicas serving the same content $c \in C$. Given access nodes request pattern $x_{i,c}$ and the replicas configuration $r_{j,c}$, we formulate the redirection problem as the **LB** optimization problem below

$$\mathbf{LB}: \quad \min \qquad F_c(\alpha) = \sum_{j \in V_R} l_{j,c}^2 / r_{j,c} \tag{1}$$

$$l_{j,c} \quad = \quad \sum_{i \in \alpha(j)} \alpha_{ij,c} x_{i,c}, \quad j \in V_R \tag{2}$$

$$\sum_{j \in \rho(i)} \alpha_{ij,c} \quad = \quad 1, \alpha_{ij,c} \geq 0, \quad i \in V_A \tag{3}$$

This is a simple convex problem with linear constraints which can be solved via standard techniques. Here we consider the well-known gradient projection method. Given an initial feasible solution $\alpha_{ij,c}(0)$, the gradient projection method solves the problem iteratively by computing a sequence $\alpha_{ij,c}(k)$ as follows:

$$\alpha_{ij,c}(k+1) = \left[\alpha_{ij,c}(k) - \delta \left(\frac{l_{j,c}}{r_{j,c}} - U_{i,c} \right) x_{i,c} \right]^+ \tag{4}$$

where $[.]^+$ denote projection on the non-negative orthant and $U_{i,c} = 1/|\rho(i)| \sum_{j \in \rho(i)} l_{j,c}/r_{j,c}$ is the average utilization of content c (among the replicas which can serve the requests of access node i) and $\delta \geq 0$ is a suitable small number. In (4) $\frac{l_{j,c}}{r_{j,c}}$ is the gradient of the objective function $F_c(\alpha)$ wrt $\alpha_{ij,c}$ and $\frac{l_{j,c}}{r_{j,c}} - U_{i,c}$ is the "projection" of the gradient as to satisfy the active constraints $\sum_{j \in \rho(i)} \alpha_{ij,c} = 1$.

Equations (4) have a simple interpretation. Observe that $\alpha_{ij,c}$ increases when $l_{j,c}/r_{j,c} - U_{j,c}$ is negative and decreases otherwise. Hence, for a given content c, the access node redirects more traffic to those replicas which are less utilized than the average replicas serving node i requests, while at the same time it diverts traffic away from the replicas which are utilized more than the average.

Since **LB** is a convex problem, the $\alpha_{ij,c}$ converge to a minimizer of (which is not necessarily unique) of $F_c(\alpha)$. At equilibrium, $\frac{l_{j,c}}{r_{j,c}} = U_{i,c}$ for all $j \in \rho(i)$ and $c \in C$. In other words, for each access node $i \in V_A$ and a given content $c \in C$ the rediction vector reaches an equilibrium when all the utilization $l_{j,c}/r_{j,c}$ of the replicas serving node i requests are equal.

The key observation is that the solution of this optimization problem, can be carried out in a distributed way. More precisely, each access node $i \in V_A$ computes iteratively its redirection vector $\alpha_{i,c} = (\alpha_{ij,c})_{j \in \rho(i)}$ via (4) using the sole load information $l_{j,c}$ from the replicas to which it is redirecting traffic (which we assume is periodically sent,

e.g., by piggybacking the load information in the messages answering the user requests).

Refined Approach The redirection scheme just presented ensures load balancing among the different servers. As previously noted, this does not necessarily imply efficient resource utilization since many servers may operate at low utilization levels. Clearly, in such a scenario, it is preferrable to remove replicas without affecting the level of users satisfaction.

To this end, we introduce a tuning parameter U_{mid} which represents a lower bound on the replica target utilization. Whenever the load falls below U_{mid} replicas take actions to direct requests away. In a distributed way this can be accomplished by modifying the **LB** problem as follows

$$\mathbf{LB}': \quad \min \qquad F'_c(\alpha) = \sum_{j \in V_R} l'^2_{j,c}/r_{j,c} \tag{5}$$

$$l'_{j,c} \quad = \quad \sum_{i \in \alpha(j)} \alpha_{ij,c} x_{i,c} + b_{j,c}, \quad j \in V_R \tag{6}$$

$$\sum_{j \in \rho(i)} \alpha_{ij,c} \quad = \quad 1, \alpha_{ij,c} \geq 0, \quad i \in V_A \tag{7}$$

where $b_{j,c} \geq 0$. The idea is to consider a different optimization problem where the actual load of $l_{j,c}$ (of underutilized replicas) are inflated by adding a suitable value $b_{j,c}$. The bias $b_{j,c}$ depends on the actual load $l_{j,c}$ as follows:

$$b_{j,c} \quad = \quad \begin{cases} 0 & \text{w.p.} \quad 1 \\ & U_{mid} < l_{j,c}/r_{j,c} \leq U_{max} \\ r_{j,c}(1 - U_{low}) & \text{w.p.} \frac{r_{j,c}U_{mid} - l_{j,c}}{r_{j,c}(U_{mid} - U_{low})} \\ & U_{low} < l_{j,c}/r_{j,c} \leq U_{mid} \\ r_{j,c}(2 - U_{low}) & \text{w.p.} \quad 1 \\ & l_{j,c}/r_{j,c} \leq U_{low} \\ r_{j,c}(2 - U_{low}) & \text{w.p.} \quad 1 \\ & U_{max} < l_{j,c}/r_{j,c}. \end{cases} \tag{8}$$

$b_{j,c} = 0$ when $l_{j,c} > U_{mid}$ and different from zero otherwise. In order to randomize the behavior of the different replicas in the range $U_{low} < l_{j,c} \leq U_{mid}$ we let the replicas add the bias with a given probability which depends on the actual load. This allows to remove synchronization effects among different replicas with the same load.

The redirection scheme for $\mathbf{LB}'$ is still computed in a distributed way by the access nodes using the gradient projection algorithm (4), the only difference being that the actual load $l_{j,c}$ is replaced by $l'_{j,c}$. This is simply achieved in practice by having the replicas advertise the *biased* load $l'_{j,c}$ rather the actual load $l_{j,c}$.

As a result, replicas whose load are below U_{mid} tend to be more and more underutilized: Since the redirection strategy tends to balance the load, replicas which advertise larger loads are less and less utilized. These replicas will end up receiving no more requests, and therefore they will be eventually removed.

V. Simulation results

In this section we report the results of a simulation-based performance evaluation aimed at assessing the effectiveness of the distributed heuristics we have proposed for replica placement and user requests redirection. Both the two heuristics have been implemented in the OPNET simulator.

Our performance evaluation has proceeded in two steps. First, we have performed experiments to quantify the advantages of the proposed dynamic replica placement heuristic with respect to the static schemes which have been proposed in the literature. We have then proceeded by thoroughly investigating the performance of our algorithms in realistic large scale networks. In particular, the experiments reported in this paper have validated the ability of the proposed scheme to exploit a proper tuning of the target utilization parameter U_{mid} to provide a fine grained control of the trade-off between number of allocated replicas, degree of utilization of the allocated replicas, distance of the users from the replica and frequency of replicas adds and removals. They have also aimed at checking the effectiveness of the redirection algorithm in performing load balancing, keeping the replicas at the desired level of utilization.

A. Comparison with static replica placement

Among the schemes introduced for the static scenarios, we have selected the greedy heuristic introduced by Qiu and al. in [2] for sake of benchmarking, as it combines very simple rules with excellent performance and has been used as a reference by many authors ([9]).

The greedy procedure takes as input a snapshot of the user requests, the set of possible replica sites V_R, the maximum number of replicas per site V_R^{max}. The output produced by the procedure is the number of replicas to be allocated, their location, and the content of each replica. The original scheme greedily adds replicas one by one, trying to maximize the user perceived quality (i.e., to minimize the sum of the distances between the users and their serving replicas). In particular, the first added replica is the one that minimizes the sum of the distances between the users access site and the replica site. The $i-th$ replica is added at a site v^i so that the set of the allocated replicas $v^1, v^2, ..., v^i$ minimizes the distance between user requests and replicas serving them. The procedure stops after having added k replicas.

To be able to compare the static greedy scheme performance to our heuristics, we had to slightly modify the greedy scheme to reflect our problem formulation. As in L. Qiu's proposal, the greedy approach adopted for sake of benchmarking introduces replicas one by one in the network. However, at each iteration it selects as site for the new replica the one that can most significantly increase the number of satisfied users requests. No limit is enforced on the number of replicas: new replicas are added till all users requests are satisfied (or no additional replica can be added without exceeding the number of replicas per site). Details can be found in [1]. We have considered two implementations of such greedy approach which differ in the events triggering a new computation of the replica placement.

The first greedy scheme ("instantaneous greedy" solution, or Greedy_inst below) re-executes the replica allocation whenever a change in the users requests occurs. The second runs periodically, every T seconds. In this case, if replicas are allocated according to user requests at the time when the algorithm is run, with no clue on future traffic demands, some users requests may not be able to be satisfied. In order to limit the occurrence of this problem, we have estimated the user requests dynamics for the upcoming time interval (of length T) by using RLS (Recursive Least Square) prediction. This allows us to estimate, based on current and past traffic, the future user requests traffic process from site i to content c.

We run the experiments using the topology in figure 2, with 24 access nodes (the white ones) and 7 service nodes (the grey ones). The thin lines represent slower links (weight 2), the thick ones faster links (weight 1). The aggregate requests are modeled as the superposition of long-tail Pareto on-off sources. To focus on the relative algorithm behavior we considered just one content, with $d_{max} = 6$, $V^R_{max} = 10$. For the distributed algorithm we set $U_{low} = 10\%$, $U_{max} = 90\%$ and two different values of U_{mid}, $U_{mid} = 10\%$ and $U_{mid} = 90\%$. These two values of U_{mid} represent the two possible extremes: in the first case, the algorithm behaves as a pure load balancing scheme; the second case is the other extreme with the target utilization almost as high as U_{max}. All other possible behaviors lie in between.

To compare the different algorithms we measured the user perceived quality (defined as the average distance between the node issuing a request and the replica serving it), and the CDN costs (average number of replicas and number of replica adds and removals).

Results for the different metrics are illustrated in Figure 1, along with the 99% confidence intervals. Replicas adds and removals are normalized to a time interval of lenght equal to 100000 time units. As expected, the static greedy algorithms generally result in a slightly lower number of replicas and user-replica distances than the dynamic algorithms. The difference in terms of average number of replicas (user-replica distance) with respect to the proposed dynamic replica placement is however quite limited, never topping 9% (5%). As expected, both greedy algorithms have instead very high reconfiguration costs as their decisions are oblivious of prior states. The more often replicas are re-allocated, the better the placement, the higher the reconfiguration costs.

The distributed heuristic perform comparably to the greedy ones in terms of number of replicas, distance to the best replica, but is able to reduce of up to three order of magnitudes the number of replicas adds and removals! The comparison of different settings of the U_{mid} threshold provides interesting results. The higher the U_{mid}, the lower both the average number of replicas AND the user-replica distance. This behavior can be explained by looking at the distributed algorithm behavior. The pure load balancing scheme ($U_{mid} = 10\%$) is characterized by a very small number of replicas adds and removals. This yields stable, slow-varying replica configurations, which however may result into a number of replicas higher than the minimum required. By contrast, with $U_{mid} = 90\%$ all the used replicas tend to be fully utilized and their number closely follows the minimum required by current requests (replicas are added and removed closely following the traffic dynamics). As a consequence, the algorithm now tends to remove and add replica more frequently, and by so doing, it is more likely to place the replicas close to the users according to the instantaneous traffic pattern. This in turn reduces the user-replica average distance. The toll to pay is in terms of a higher number of replicas added/removed. This number is however order of magnitudes lower than in the greedy static scenarios.

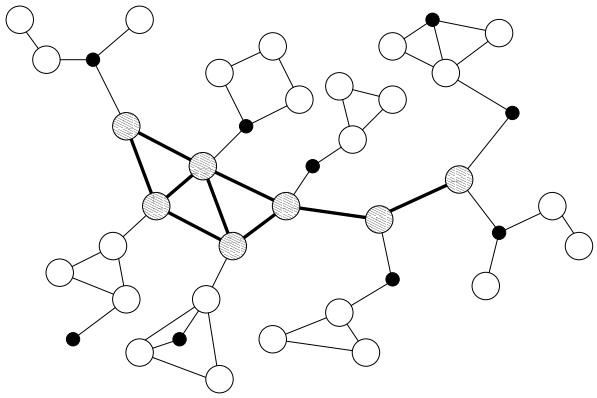

Figure 2. NETWORK TOPOLOGY USED IN THE EXAMPLE

B. Distributed Heuristics Evaluation

We now turn our attention to the dynamic behavior of the proposed distributed algorithms. In this set of simulations we considered a more realistic scenario based on the topology of the AT&T backbone (see Figure 3 and reference [10]) which comprises 184 access nodes and 115 service nodes. The requests have been modeled as the superposition of indipendent on-off Pareto processes.

In tables I - II, we summarized the relevant metrics along with the 99% confidence intervals for different value of U_{mid} and d_{max}. Values for the number of replicas adds and removals are again normalized to an interval of 1000 time units.

The behavior for the different values of U_{mid} confirm our previous observations: lower values of U_{mid} yield more stable replica configurations which in turn result into a larger number of replicas and higher user-replicas distances.

Comparing the two tables, we observe that, with smaller d_{max}, the average number of used replicas increases while the average distance decreases. This reflects the need to place more replicas to meet the stricter constraint on the maximum user-replice distance. Such replicas are necessary placed closer to the users, resulting in lower user-request distances.

Figure 4 illustrates a typical sample path behavior. Given one content, the figure compares the aggregated user requests over time (the lower curve), *i.e.*, $\sum_{i \in V_A} x_{i,c}$ with the maximum amount of requests which can be satisfactorly served by the allocated replicas, *i.e.*, $\sum_{j \in V_R} r_{j,c}$ for different value of U_{mid}. If the two curves are closer it means that the heuristic

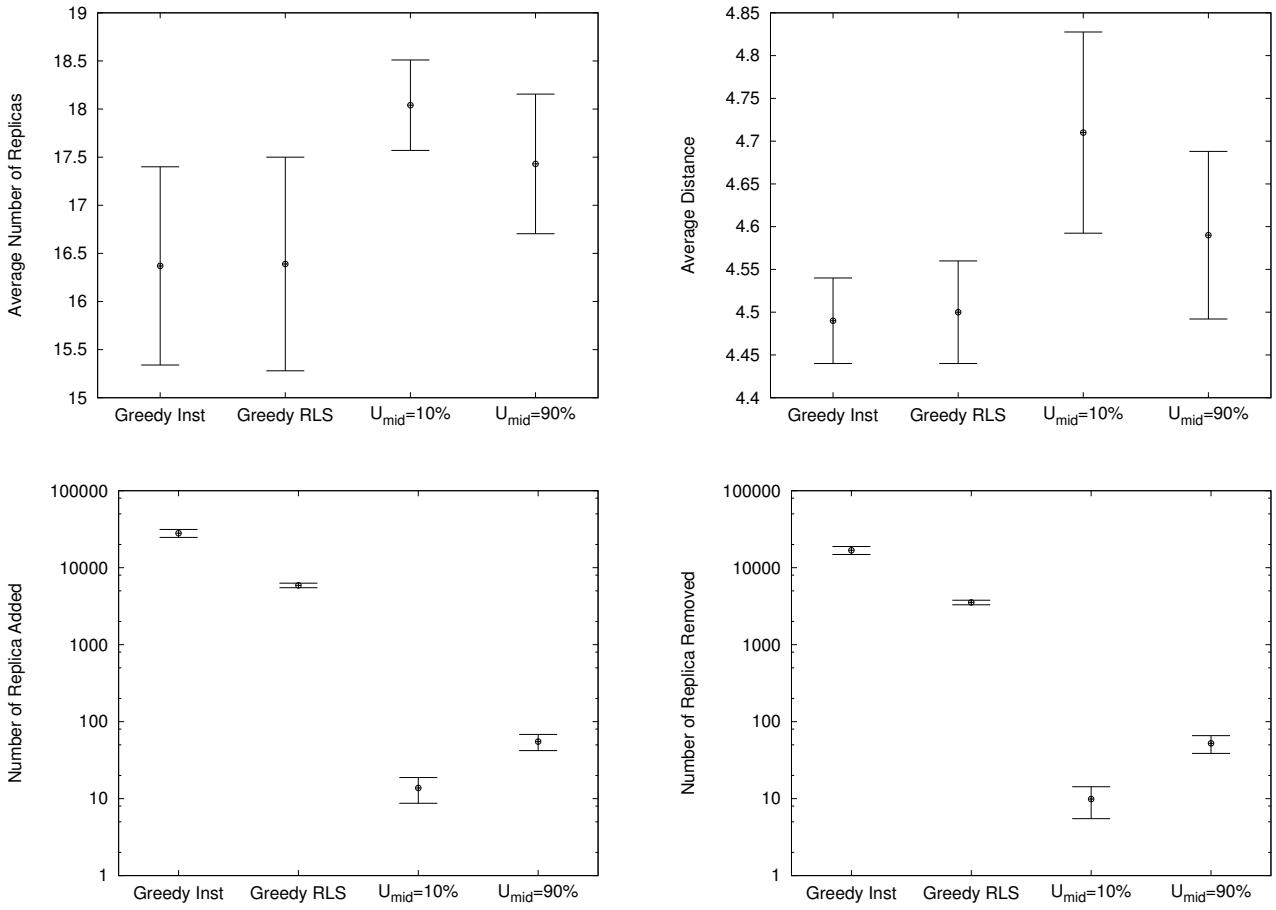

Figure 1. SIMULATION RESULTS one content and $d_{\max} = 6$

closely follows the traffic dynamics only allocating the minimum quantity of replicas needed to satisfy the users needs. This is exactly what happens as U_{mid} increases: the replica placement algorithm and the redirection schemes follow more closely the user demand, resulting in curves which almost overlap with the users requests. At the same time, the number of replica cloning and removal, and hence the corresponding cost, is kept reasonably low.

Finally, in Figure 5 we show the replicas load distibution observed over a simulation run executed on the topology shown in Fig. 2. Traffic is Poisson-distributed. We plot vertical lines corresponding to the values of $U_{low} = 8\%$, $U_{mid} = 75\%$ and $U_{max} = 92\%$. The figure shows how well our redirection algorithm allows (in a completely distributed and localized way) to achieve the desired target utilization behavior for a replica (in this case $75\% \leq l_{j,c} \leq 92\%$). As soon as traffic dynamics bring the replicas' load outside the desired range immediate action is taken: either by adding new replicas, or by first offloading and then removing some of the existing replicas.

Figure 3. AT&T NETWORK TOPOLOGY (from paper [11])

U_{mid}	46%	60%	90%
avg. distance	13.01 ± 0.163	12.98 ± 0.187	12.78 ± 0.093
replica add	0.19 ± 0.224	0.54 ± 0.234	3.87 ± 0.915
replica del	0.15 ± 0.162	0.48 ± 0.334	3.86 ± 0.956
replicas	54.25 ± 8.882	49.10 ± 2.349	45.84 ± 1.592

Table I
AT&T, $d_{max} = 15$

U_{mid}	46%	60%	90%
avg. distance	14.19 ± 0.353	14.18 ± 0.370	13.26 ± 0.150
replica add	0.09 ± 0.138	0.09 ± 0.138	3.31 ± 0.353
replica del	0.00 ± 0.000	0.00 ± 0.000	3.31 ± 0.496
replicas	47.79 ± 2.005	47.79 ± 2.005	45.22 ± 1.647

Table II
AT&T, $d_{max} = 200$

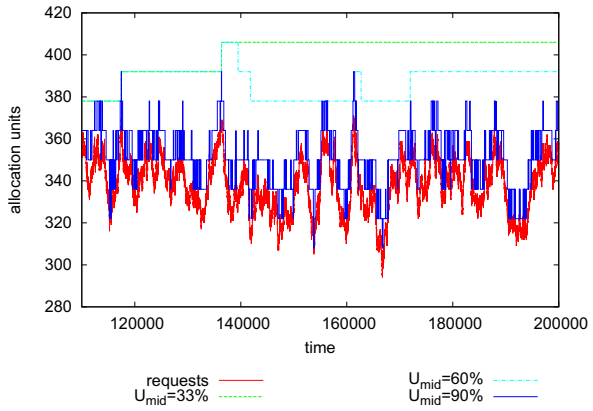

Figure 4. Simulation traces

VI. Conclusions

In this paper we have jointly addressed dynamic replica placement and user requests redirection in Content Delivery Networks. The solutions we have proposed are fully distributed and localized and allow to minimize the costs for replica placement, maintenance, replicas adds/removals while being able to satisfy all users requests and to keep allocated replicas above a target level of utilization.

Extensive OPNET simulations have allowed us to prove the effectiveness of our distributed heuristics. In particular simulations we have performed have shown that:

- The proposed solution improves over static replica placement. To achieve comparable performance in terms of number of allocated replicas, distance between the user and the serving replicas, static schemes have to be re-executed frequently, resulting in a number of replica adds and removals that can be three orders of magnitude higher than in our scheme.
- The heuristics perform well in terms of number of replicas, replicas utilization, distance between the access site and the serving replica, frequency of configuration changes, when varying the traffic load, the network topology, the type of traffic (Poisson, long tail). In particular, the redirection scheme is very effective in performing load balancing, keeping replicas within the specified interval of utilization. By setting different targets levels U_{mid} of utilization of the replicas we can strictly control the CDN network operations and trade-off between number of replicas, replicas utilization and frequency of changes in the replica placement.

Overall, the proposed schemes have proven to be a promising low complexity, low overhead tool to control the CDN operations.

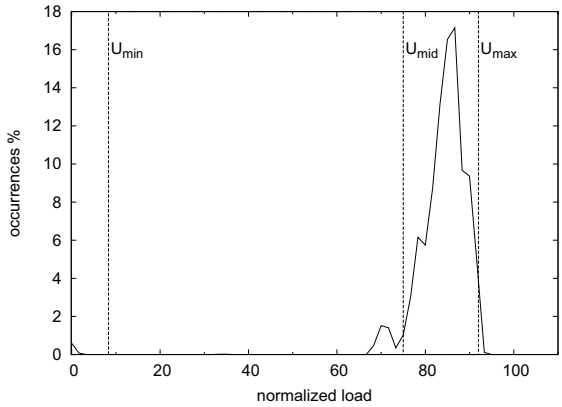

Figure 5. Behaviour of the distributed load balancing

References

[1] F. Lo Presti, C. Petrioli, and C. Vicari, "Dynamic replica placement in content delivery networks," in *Proceedings of MASCOTS 05*, September 2005.

[2] L. Qiu and al ., "On the placement of web server replicas," in *Proceedings of IEEE INFOCOM 2001*, Anchorage, AK, April 22–26 2001, pp. 1587–1596.

[3] S. Jamin and al., "Constrained mirror placement on the internet," in *Proceedings of IEEE INFOCOM 2001*, Anchorage, AK, April 22–26 2001, pp. 31–40.

[4] P. Radoslavov and al., "Topology-informed internet replica placement," *Proceedings of WCW'01*, June 20–22 2001.

[5] M. Rabinovich and A. Aggarwal, "RaDaR: a scalable architecture for a global Web hosting service," *Elsevier Computer Networks*, vol. 31, no. 11–16, pp. 1545–1561, 1999.

[6] Y. Chen, R. Katz, and J. Kubiatowicz, "Dynamic replica placement for scalable content delivery," in *International Workshop on Peer-to-Peer Systems, IPTPS 2002*, Cambridge, MA, March 7–8 2002.

[7] N. Bartolini, F. Lo Presti, and C. Petrioli, "Dynamic replica placement and user request redirection in content delivery networks," in *IEEE International Conference on Communications, 2005 (ICC 2005)*, May 2005.

[8] J. Gossa, J. M. Pierson, and L. Brunie, "Fredi: Flexible replica displacer," in *Proceeding of the International Conference on Networking, International Conference on Systems and Intenational Conference on Mobile Communications and Learning Technologies (ICNICONSMCL-06)*, 2006.

[9] M. Karlsson and M. Mahalingam, "Do we need replica placement algorithms in content delivery networks," in *7th International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[10] "Project rocketfuel." [Online]. Available: http://www.cs.washington.edu/research/networking/rocketfuel

[11] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Transactions on Networking*, no. 1, pp. 2–16, February 2004.

[12] M. Karlsson, C. Karamanolis, and M. Mahalingam, "A unified framework for evaluating replica placement algorithms," *Technical Report HPL-2002, Hewlett Packard Laboratories*, 2002.

[13] N. Bartolini, F. Lo Presti, and C. Petrioli, "Optimal dynamic replica placement in Content Delivery Networks," in *Proceedings of ICON 2003*, Sydney, Australia, September 28–October 1 2003, pp. 125–130.

[14] "Opnet university program." [Online]. Available: http://www.opnet.com/services/university