Optimal Dynamic Replica Placement in Content Delivery Networks

Novella Bartolini[§] Francesco Lo Presti[‡] Chiara Petrioli[§]

[§]Università di Roma "La Sapienza"
Dipartimento di Informatica
Via Salaria 113, 00198 Roma, Italy

[‡] Università dell'Aquila
Dipartimento di Informatica
Via Vetoio, 67010 Coppito (AQ), Italy

Abstract— **Content Delivery Networks (CDN) design entails the placement of server replicas to bring the content close to the users, together with an efficient and content aware request routing. In this paper we address the problem of *dynamic* replica placement to account for users demand variability while optimizing the costs paid by a CDN provider and the overall performance of the distributed replica servers architecture. We formulate the dynamic replica placement problem as a Semi Markov Decision Process accounting for the traffic conditions, the users level of satisfaction, as well as the costs paid to install, maintain or remove a replica from a site. The proposed model applies to general network topologies and considers realistic constraints on network and servers capacity. The optimal strategy derived by means of the decision model provides the ground for designing a centralized heuristic and is used as a benchmark for the heuristic evaluation. Simulation results show that the proposed heuristic has a behavior close to that of the optimal strategy and achieves very good performance in terms of low average distance from a user to the serving replica, low average number of replicas and high probability of being able to serve a request.**

Keywords— **Dynamic Replica Placement, Content Delivery Networks, Semi Markov Decision Processes**

I. Introduction

The commercial success of the Internet has paved the way for the birth of Content Delivery Networks (CDN) providers. CDN providers are companies devoted to hosting in their servers the content of third-party content providers, to mirroring or replicating such contents on several servers spread over the world, and to transparently redirecting the customers requests to the 'best replica' (e.g. the closest replica, or the one from which the customer would access content at the lowest latency). Designing a complete solution for CDN therefore requires addressing a number of technical issues: which kind of content should be hosted (if any) at a given CDN server (replica placement), which is the 'best replica' for a given customer, which mechanisms are used to transparently redirect the user to such replica. In this paper we focus on the first problem: replica placement.

The majority of the schemes presented in the literature tackle the problem of static replica placement that can be formulated as follows. Given a network topology, a set of CDN servers and a given request traffic pattern, decide where content has to be replicated so that some objective function is optimized while meeting constraints on the system resources. The solutions so far proposed typically try to either maximize the user perceived quality given an existing infrastructure, or to minimize the CDN infrastructure cost while meeting a specified user perceived performance. Examples of constraints taken into account are limits on the servers storage, on the servers sustainable load, on the maximum delay tolerable by the users etc. A thorough survey of the different objective functions and constraints considered in the literature can be found in [6].

This work has been funded by the WEB-MINDS project supported by the Italian MIUR under the FIRB program and by the POLLENS project supported by ITEA.

For the static case, simple efficient greedy solutions have been proposed in [8], [5] and [10]. In [8] Qiu et al. formulate the static replica placement problem as a minimum K median problem, in which K replicas have to be selected so that the sum of the distances between the users and their 'best replica' is minimized. A simple greedy heuristic is shown to have performance within 50% of the optimal strategy. In [5] and [10] 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. The rationale is that such nodes are likely to be in strategic places, closest (on average) to all other nodes, and therefore suitable for replica location. In [10] a performance evaluation based on real-world router-level topology shows that the fan-out based heuristic has behavior close to the greedy heuristic in terms of the average client latency.

All these solutions lack in considering the dynamics of the system (e.g. changes in the requests traffic pattern, network topology, replica sites). As a worst-case replica placement would result extremely inefficient, this is indeed an important practical issue that has to be tackled. A simple approach could consist in periodically executing the static replica placement algorithms to react to system dynamics. However this approach has a twofold drawback: it may react slowly to the system dynamics (depending on the period between two different executions of the algorithm) and selects the 'optimal' replica placement irrespective of the current configuration, possibly leading to significant reconfiguration costs.

In this paper we propose a different (and more natural) approach. We introduce a dynamic allocation strategy which explicitly takes into account the system dynamics as well as the costs of modifying the replica placement. The contributions of the paper are twofold. By assuming the users requests dynamics to obey to a Markovian model we first formulate the dynamic replica placement problem as a Markovian decision process. Albeit this model may not accurately capture the user dynamics and can be numerically solved only for limited sized CDNs, it allows us to identify an optimal policy for dynamic replica placement that can be used as a benchmark for heuristics evaluation and provides insights on how allocation and deallocation should be performed. Based on the findings obtained through the analytical model we derive and evaluate a centralized heuristic which allocates and deallocates replicas to reflect the requests traffic dynamics, the costs of adding, deleting and maintaining replicas, the servers load and storage limits, and the requirements on the maximum distance of the users from the 'best replica'. The heuristic performance evaluation shows that its behavior is very close to that of the optimal placement strategy, and that the heuristic achieves very good performance in terms of the low average number of replicas, the low user-replica average distance and the small number of requests that cannot be served.

A few previous papers (e.g. [9] and [3]) have addressed the problem of dynamic replica placement. However, the proposed schemes are embedded in specific architectures for performing requests redirection and computing the best replicas. No framework is provided for identifying the optimal strategy, and to quantify the solutions performance with respect to the optimum. In RaDar [9] a threshold based heuristic is proposed to replicate, migrate and delete replicas in response to system dynamics. The overall proposed solution combines dynamic replica allocation with servers load aware redirection to the best replica to achieve low average users perceived latency while empirically balancing the load among the CDN servers. No limits on the servers storage and on the maximum users perceived latency are explicitly enforced. In [3] two schemes designed for the Tapestry architecture are presented. The idea is that upon a request for content the access point neighborhood in the overlay network is searched. If there is a server hosting a replica of the requested content within a maximum distance from the user, and such server is not overloaded, the request will be redirected to this server (or to the closest server if multiple servers meet such constraints). Otherwise a new replica is added to meet the user request. Two variants are introduced depending on the neighborhood of the overlay network which is searched for replicas, and on the scheme used to select the best location for the new replica. Though the ideas presented in the paper appear promising they are tightly coupled with the Tapestry architecture, and the approach does not explicitly account for neither the costs of reconfiguration nor for possible servers storage limits. Finally, no information is provided in [3] on the rule to remove replicas, making it hard to compare with this approach.

The paper is organized as follows. In sections II and III the dynamic replica placement problem is formally formulated and discussed, and the semi Markov decision model used to derive the optimal strategy is described. In section IV a centralized heuristic is presented, followed by its performance evaluation (section V). A summary concludes the paper in section VI.

II. Problem statement

We model the Internet network topology as a weighted undirected graph $G = \{V, E\}$. The vertex set V is the set of network nodes, while each edge in the set E represents a physical network link and is labeled with some kind of distance metric, e.g. the number of hops between the endpoints or a more complex function that takes into account the available bandwidth, giving lower cost to the backbone links than to the low speed access links. We identify two subsets V_A and V_R of the set of network nodes V. V_A is the set of CDN access nodes where the requests generated by the users enter the core CDN network. V_R is the set of nodes in the core network where one or more content replica servers can be placed (called sites in the following). Figure 5 (c) shows a possible 40 nodes hierarchical transit stub network topology obtained by running the `gt-itm` topology generator [1]. The white circles represent the access nodes, the grey big circles the sites that can host replicas, and the small black circles nodes only used for sake of routing. Thin and thick links reflect the low or high bandwidth of the links.

We assume that C content providers exploit the hosting service of the CDN. Customers entering the CDN through an access node in V_A can therefore issue requests for one of C sets of contents, and replicas of some of the C contents can be placed in each site in V_R. Requests entering the CDN are measured in units of aggregate requests. No more than V_A^{MAX} units of aggregate requests can be generated by an access node (to model the limited access link bandwidth). Requests for a given content are served by a suitable replica. To model user satisfaction, we assume that user requests cannot be served by a replica at a distance above a given threshold d_{max}. Users requests are redirected to the best available replica. This can be accomplished by several means, *i.e.*, anycast. We assume that each replica can serve up to K unit of aggregate request for that content (replica service capacity limit). No more than V_R^{MAX} replicas can be hosted at a given site (site resource limit).

We describe a given configuration of requests and replicas by means of a state vector $\mathbf{x}$ of size $C(|V_A| + |V_R|)$:
$\mathbf{x} = (\mathbf{a}, \mathbf{r}) = (a_1^1, a_2^1, \ldots, a_{|V_A|}^1, a_1^2, \ldots, a_{|V_A|}^2, \ldots, a_{|V_A|}^{\mathsf{C}},$
$r_1^1, r_2^1, \ldots, r_{|V_R|}^1, r_1^2, \ldots, r_{|V_R|}^2, \ldots, r_{|V_R|}^{\mathsf{C}})$
in which the variable a_i^c represents the number of request units for a content $c \in \{1, \ldots, C\}$ at node $i \in V_A$, and r_j^c is the number of replicas of content $c \in \{1, \ldots, C\}$ placed at site $j \in V_R$.

We associate to each state a cost - paid per unit of time - which is the sum of a cost derived from the users perceived quality (users to replica distance, number of unsatisfied requests) and of the CDN infrastructure costs for hosting and maintaining replicas.

We measure users perceived quality by means of a function $A(\mathbf{x})$. This is given by the sum over all users requests of the distance between the access node where the request is originated and the replica serving it. Since requests are served by the best available replica (i.e. the closest according to to links metric), the redirection itself requires the solution of a minimum cost matching problem between the users requests and the available replicas, with the distance between access nodes and service site as cost of the matching (assuming infinite cost whenever the distance exceeds the threshold d_{max}). The solution of this problem yields the redirection scheme (which request is served by which replica) and the associated distance (cost).

A replica maintenance cost $M(\mathbf{x})$ is used to model the costs of hosting replicas and keeping them up to date. We use a simple proportional cost model $M(\mathbf{x}) = C_{Maint} \sum_{j \in V_R, c=1,\ldots,C} r_j^c$, where C_{Maint} is a suitable constant.

Two other costs C^+ and C^- are paid by the CDN provider when dynamically adjusting the number of replicated servers, and are associated to the decision to add or remove a replica respectively.

The minimization of the long run costs described above enables a decision making criterion that can be used to formulate dynamic replica placement strategies. Given a state, a cost function associated to it and the costs of replicating and deleting replicas, identify a strategy which dynamically allocates and deallocates replicas in response to users demand variations so that the overall cost is minimized while meeting the constraints on the replica service capacity and site resources. In the following we introduce a Markov decision process to derive the optimal strategy for solving this problem as well as a centralized scalable heuristic.

III. A Markov decision process for dynamic replica placement

The state of the Semi Markov Decision Process (SMDP) is formulated as in the previous section II by a vector $\mathbf{x}$ of size

$C(|V_A|+|V_R|)$:
$\mathbf{x}=(\mathbf{a},\mathbf{r})=(a_1^1,a_2^1,\ldots,a_{|V_A|}^1,a_1^2,\ldots,a_{|V_A|}^2,\ldots,a_{|V_A|}^{\mathsf{C}},$
$r_1^1,r_2^1,\ldots,r_{|V_R|}^1,r_1^2,\ldots,r_{|V_R|}^2,\ldots,r_{|V_R|}^{\mathsf{C}})$

The states space Λ is then defined as:

$$\Lambda=\{\mathbf{x}=(\mathbf{a},\mathbf{r}):\sum_{k=1}^{C}a_i^k\leq V_A^{\mathsf{MAX}};\sum_{k=1}^{C}r_j^k\leq V_R^{\mathsf{MAX}};\\ a_i^k,r_j^k\geq 0,\forall i\in V_A,j\in V_R\}.$$

Since the population of the described model is an aggregate figure of the requests traffic, the process dynamics is determined by changes in the average units of requests at an access site for a given content. We model the aggregate request at site $i\in V_A$ for a content k as a birth death process with birth-rate λ_i^k and death-rate μ_i^k, respectively.

When the system is in state $\mathbf{x}$, a state dependent decision can be made: a new replica of a given content can be placed at or removed by a site or the system can be left as it is.

The action space $\mathcal{D}$ can be expressed by

$$\mathcal{D}=\Big\{(d_1^{1+},d_2^{1+},\ldots,d_{|V_R|}^{1+},\ldots,d_1^{\mathsf{C}+},\ldots,d_{|V_R|}^{\mathsf{C}+},d_1^{1-},\\ d_2^{1-},\ldots,d_{|V_R|}^{1-},\ldots,d_1^{\mathsf{C}-},\ldots,d_{|V_R|}^{\mathsf{C}-}),d_i^{kx}\in\{0,1\},\\ \sum_{\forall i,k,x}d_i^{kx}\leq 1;i=1,\ldots,|V_R|;x=+/-\}.$$

The indicator $d_i^{k+}=1$ represents the decision to add a replica of content k in site $i\in V_R$, while $d_i^{k-}=1$ stands for the decision to remove a replica of content k from site i. If all the indicators are null, the corresponding decision is to leave the replica placement as it is. For simplicity only one replica can be placed or removed at each decision time.

The action space is actually a state-dependent subset of $\mathcal{D}$ where a decision to add a replica is allowed only if the number of replicas hosted at the site is less than the maximum number of replicas V_R^{MAX}, and a decision to remove a replica can be made only if at least one replica is actually hosted in the considered site.

If the system is in state $\mathbf{x}=(\mathbf{a},\mathbf{r})\in\Lambda$ and the action $\mathbf{d}\in\mathcal{D}$ is chosen, then the dwell time of the state $\mathbf{x}$ is $\tau(\mathbf{x},\mathbf{d})$ where

$$\tau(\mathbf{x},\mathbf{d})=\left[\sum_{i=1}^{|V_A|}\sum_{k=1}^{C}(\lambda_i^k+a_i^k\cdot\mu_i^k)\right]^{-1}. \quad (1)$$

The transitions that may occur in this model from an initial state $\mathbf{x}$ to a final state $\mathbf{y}$ can be due to an increasing aggregate request, in the form of a birth, or to a decreasing request, in the form of a death in the underlying multidimensional birth and death process. The transition probability $p_{\mathbf{xy}}^{\mathbf{d}}$ from the state $\mathbf{x}=(\mathbf{x_A},\mathbf{x_R})$ to any state $\mathbf{y}=(\mathbf{y_A},\mathbf{y_R})\in\Lambda$ under the decision $\mathbf{d}$, takes one of the following expressions, where $\mathbf{e}_i^c$ is an identity vector with unary element in position $(c\cdot|V_A|+i)$.

Transitions due to an arrival of a unit of aggregate request for content c at site i:
to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}+\mathbf{e}_i^c,\mathbf{x_R}+\mathbf{e}_j^k)$
with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=\lambda_i^c\cdot d_j^{k+}\tau(\mathbf{x},\mathbf{d}); \quad (2)$$

to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}+\mathbf{e}_i^c,\mathbf{x_R}-\mathbf{e}_j^k)$
with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=\lambda_i^c\cdot d_j^{k-}\tau(\mathbf{x},\mathbf{d}); \quad (3)$$

to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}+\mathbf{e}_i^c,\mathbf{x_R})$
with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=\lambda_i^c\cdot\tau(\mathbf{x},\mathbf{d})\cdot[1-\sum_{j=1}^{|V_R|}\sum_{k=1}^{C}(d_j^{k+}+d_j^{k-})]. \quad (4)$$

Transition due to a departure of a unit of aggregate request for content c at site i:
to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}-\mathbf{e}_i^c,\mathbf{x_R}+\mathbf{e}_j^k)$ with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=(\mathbf{x_A})_i^c\cdot\mu_i^c\cdot d_j^{k+}\cdot\tau(\mathbf{x},\mathbf{d}); \quad (5)$$

to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}-\mathbf{e}_i^c,\mathbf{x_R}-\mathbf{e}_j^k)$ with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=(\mathbf{x_A})_i^c\cdot\mu_i^c\cdot d_j^{k-}\cdot\tau(\mathbf{x},\mathbf{d}); \quad (6)$$

to state $(\mathbf{y_A},\mathbf{y_R})=(\mathbf{x_A}-\mathbf{e}_i^c,\mathbf{x_R})$ with probability

$$p_{\mathbf{xy}}^{\mathbf{d}}=(\mathbf{x_A})_i^c\cdot\mu_i^c\cdot\tau(\mathbf{x},\mathbf{d})\cdot[1-\sum_{j=1}^{|V_R|}\sum_{k=1}^{C}(d_j^{k+}+d_j^{k-})]. \quad (7)$$

The transitions that are not considered in this list have probability 0.

In order to create a decision criterion for the described model, an objective function is formulated. The state-related costs $A(\mathbf{x})$ and $M(\mathbf{x})$ introduced in section II are paid per unit of time as long as the system persists in the considered state $\mathbf{x}$. The costs C^+ and C^- are instead transition-related and are only paid when a replica is actually added or removed, i.e. when the corresponding transition occurs.

Therefore a non-uniform cost function can be formulated as

$$r(\mathbf{x},\mathbf{d})=[A(\mathbf{x})+M(\mathbf{x})]\cdot\tau(\mathbf{x},\mathbf{d})+\\ +\sum_{j=1}^{|V_R|}\sum_{k=1}^{C}(d_j^{k+}C^+ + d_j^{k+}C^-)\}.$$

The uniformization technique [2], [7], [11] transforms the original SMDP with non identical transition times into an equivalent continuous-time Markov process in which the transition epochs are generated by a Poisson process at uniform rate. The transitions from state to state are described by a (discrete time) Markov chain that allows for fictitious transitions from a state to itself. The uniformized Markov process is probabilistically identical to the non uniform model. The theory of discrete Markov processes can then be used to analyze the discrete-time embedded Markov chain of the uniformized model.

A uniform rate Γ can be taken as an upper bound on the total outgoing rate from each state thus obtaining a continuous time, uniform process with rate $1/\Gamma$. The following definition of Γ fits our needs.

$$\Gamma=\sum_{i=1}^{|V_A|}\sum_{k=1}^{C}[\lambda_i^k+V_A^{\mathsf{MAX}}\cdot\mu_i^k].$$

The transition probabilities of the uniformized process are formulated as in equations (2 - 7), substituting the non uniform dwell time $\tau(\mathbf{x}, \mathbf{d})$ defined in equation (1) with the uniform dwell time $1/\Gamma$, and adding dummy transitions from each state to itself: $\mathbf{y} = \mathbf{x}$

$$\tilde{p}^{\mathbf{d}}_{\mathbf{xy}} = \frac{1}{\Gamma} \cdot [\Gamma - \sum_{i=1}^{|V_A|} \sum_{k=1}^{C} (\lambda_i^k + \mu_i^k \cdot (\mathbf{x_A})_i^k)].$$

The cost function is uniformized as well, obtaining the following formulation of $\tilde{r}(\mathbf{x}, \mathbf{d})$:

$$\tilde{r}(\mathbf{x}, \mathbf{d}) = \frac{r(\mathbf{x}, \mathbf{d})}{\tau(\mathbf{x}, \mathbf{d}) \cdot \Gamma} = \frac{1}{\Gamma}\{[A(\mathbf{x}) + M(\mathbf{x})] + \\ + \frac{1}{\tau(\mathbf{x}, \mathbf{d})} \cdot \sum_{j=1}^{|V_R|} \sum_{k=1}^{C} (d_j^{k+} C^+ + d_j^{k+} C^-)\}.$$

An optimal solution can be expressed through a decision variable $\pi_{\mathbf{xd}}$ that represents the probability for the system to be in state $\mathbf{x}$ and taking the decision $\mathbf{d}$.

The Linear Programming (LP) formulation associated with our SMDP minimizing the cost paid in the long-run execution is given by:

$$\begin{array}{l} \text{Minimize} \\ \sum_{(\mathbf{x},\mathbf{d}) \in \mathcal{S}} \tilde{r}(\mathbf{x}, \mathbf{d}) \cdot \pi_{\mathbf{x},\mathbf{d}} \\ \\ \text{constrained to} \\ \pi_{\mathbf{xd}} \geq 0 \qquad\qquad (\mathbf{x}, \mathbf{d}) \in \mathcal{S} \\ \sum_{(\mathbf{x},\mathbf{d}) \in \mathcal{S}} \pi_{\mathbf{xd}} = 1 \\ \sum_{\mathbf{d} \in \mathcal{B}} \pi_{\mathbf{jd}} = \sum_{(\mathbf{x},\mathbf{d}) \in \mathcal{S}} \tilde{p}^{\mathbf{d}}_{\mathbf{xj}} \pi_{\mathbf{xd}} \qquad \mathbf{j} \in \Lambda. \end{array} \tag{8}$$

where $\mathcal{S}$ is the finite set of all feasible couples of vectors of the kind (*state, decision*).

The problem defined in (8) can be solved by means of commonly known methods of the operations research [4]. We used the simplex method with sparse matrix support.

IV. Heuristic

The solution to the optimization problem (8) is too computationally intensive but in the simplest scenarios. Therefore, in general, it is not feasible to compute the optimal policy. Here we propose an heuristic to decide the action $\mathbf{d} \in \mathcal{D}$ to take upon transitions on the request access vector $\mathbf{a}$. The heuristic has been derived by closely studying how the optimal policy behaved in our experiments. In particular, we considered the case where the cost function imposes - in decreasing order - the following priorities to the resulting policy: (1) being able to serve user requests; (2) minimizing the number of replicas; (3) minimizing the distance between users and replicas. This was accomplished by setting the cost function parameters as follows: $C_{maint} >> \max_{e \in E} \ell(e)$, with $\ell(e)$ denoting the weight associated with link e, $C^+ = C^- = 0$. With this choice, we expected the optimal policy to use the minimum number of replicas to serve all existing requests leaving at the same time enough spare capacity to accommodate for requests increases. In our experiments, indeed, we observed that the optimal placement policy proactively replicates content in order to guarantee its availability in case of future requests increases. At the same time, to minimize the number of replicas, it detects and removes replicas which are not needed to serve either current requests or any possible unitary increase of them.

```
1. d =do nothing;
2. while ( TRUE ) {
3.     wait for a change in a; take action d;
4.     if ( enough_replica_on_increase( (a,r ) )
5.         d=remove_replica();
6.     else
7.         d=add_replica();
8. }
```

Fig. 1. Replica Placement Algorithm.

```
1. boolean enough_replica_on_increase( state (a,r ) ) {
2.     for any c = 1, ..., C and i ∈ V_A
3.         if ( !enough_replica( (a + e_i^c, r) ) return FALSE;
4.     return TRUE;
5. }
```

Fig. 2. enough_replica_on_increase().

The algorithm we propose, shown in Figure 1, mimics as close as possible this behavior. At each step, the algorithm determines first whether the current replica configuration $\mathbf{r}$ can accommodate any possible increase in user requests $\mathbf{a}$. This is accomplished via the function enough_replica_on_increase() (see Figure 2). In case that any possible increase in user requests can be accommodated by $\mathbf{r}$ then the algorithm considers whether it is possible to remove a replica (remove_replica()); otherwise it tries to find a site where to add a replica (add_replica()). (Observe that to mimic the behavior of the Markovian Decision Process, actions are decided in a given state but only taken in correspondence of the next transition.)

The function enough_replica_on_increase((a, r)) returns TRUE if any possible increase in $\mathbf{a}$ can be served by the current replica configuration $\mathbf{r}$ and FALSE otherwise. To this end, it uses the function enough_replica((a, r)) which determines whether a given users access requests $\mathbf{a}$ can be served by the set of replicas $\mathbf{r}$. (The function enough_replica() itself is computed by solving a minimum matching problem between users requests and the available replicas from the solution of which we can determine whether all request in $\mathbf{a}$ can be served by $\mathbf{r}$.)

add_replica() is called to determine content and location for a

```
1.  action add_replica ( ) {
2.      find I = {(i, c) i ∈ V_A, c = 1, ..., C | !enough_replica((a + e_i^c, r))}
3.      for any j ∈ V_R, c = 1, ..., C
4.          find J(j, c) = {(i, c) ∈ I | enough_replica(a + e_i^c, r + e_j^c),
                (a + e_i^c, r + e_j^c) ∈ Λ}
5.      if (max_(j,c) |J(j, c)| > 0)
6.          (j*, c*) = argmax |J(j, c)|;
7.          d=place replica content c* in site j*;
8.      else
9.          d=do nothing;
10.     return d;
11. }
```

Fig. 3. Algorithm for Deciding where to Place a New Replica.

1. action remove_replica ( ) {
2. find $U = \{(j,c)\ j \in V_R, c = 1,\ldots,C \mid \exists i \in V_A$ enough_replica $((\mathbf{a}+\mathbf{e}_i^c, \mathbf{r}))$ AND !enough_replica $((\mathbf{a}+\mathbf{e}_i^c, \mathbf{r}-\mathbf{e}_j^c))$, $(\mathbf{a}+\mathbf{e}_i^c, \mathbf{r}-\mathbf{e}_j^c) \in \Lambda\}$;
3. find $J = \{(j,c)\ j \in V_R, c = 1,\ldots,C\ (j,c) \notin U \mid$ enough_replica$((\mathbf{a}, \mathbf{r}-\mathbf{e}_j^c))$, $(\mathbf{a}, \mathbf{r}-\mathbf{e}_j^c) \in \Lambda\}$;
4. if ($|J| > 0$)
5. $(j^*, c^*) = \text{argmin}_{(j,c)\in J}|S_{j^*}|$, $S_j = \{i,\ i \in V_A \mid distance(i,j) \leq d_{Max}\}$;
6. $\mathbf{d}$=remove replica c^* in site j^*;
7. else
8. $\mathbf{d}$=do nothing;
9. return $\mathbf{d}$;
10. }

Fig. 4. ALGORITHM FOR DECIDING WHICH REPLICA TO REMOVE.

new replica. To this end it first identifies which requests increase would require additional replicas. This is accomplished in line 2 by determining the set I of the pairs (i,c) such that an increase of requests for content c at node i cannot be served by $\mathbf{r}$ (line 2). It then computes the sets $J(j,c) \subseteq I$ of users requests increment that could be served by an additional replica of content c in site j (line 3). If not all $J(j,c)$ are empty , the content and location of the additional replica is then chosen by finding the site j^* and content c^* which maximizes $|J(j,c)|$ (line 7). This to maximize the probability of being able to satisfy a request increase.

remove_replica() is called to determine whether to remove a replica. To this end, first it identifies the set U of replicas which should not be removed as they would be needed to serve an increase in users requests (line 2). Then, it determines, among the remaining replicas, the set J of the candidates for possible removal, *i.e.*, all those replicas which are not used to serve current requests (line 3). Among these, it chooses to remove a replica from a node j which serves the smallest set of access nodes (line 5). Choosing the replica which serves the smallest population should minimize the likelihood to remove a replica which is going to be added soon again.

V. NUMERICAL RESULTS

In this section we evaluate and compare the performance of the proposed replica placement algorithms. For the optimal replica placement, we computed the optimal policy by solving the Markov decision model with MATLAB. For the heuristic, we wrote a simulator in C. The network topologies used in our experiments are reported in Figure 5. The thin lines denote slower links (with a weight of 2), the thick ones faster links (with a weight of 1). We considered the small topologies to assess the heuristic effectiveness with respect to the optimal policy. (Indeed only for such smaller cases we were able to compute the optimal policy.) The larger topology, generated with the `gt-itm` topology generator, was used to test the heuristic in a more realistic setting. In all simulations, we used the following parameters: $K = 2$, $V_A^{\mathsf{MAX}} = 2$, $\lambda_i^c = 1/C$ and $\mu_i^c = 1$. Moreover, we set $C_{maint} = 1000$, $C^+ = C^- = 0$. The values of d_{max}, of V_R^{MAX} and the number of contents C were varied in the different experiments.

The results for the small topologies are reported in table I and II, respectively. In these experiments we set $V_R^{\mathsf{MAX}} = 1$ (at most one replica per site) and $d_{max} = 3$. For the comparison, rather than reporting the cost function values, we separately reported the values of the three major factors which contribute to the cost, namely, the average distance from a request to the serving replica, the average number of replicas, and the fraction of requests which were not served (either because no such replica existed or because the closest available replica was further away than d_{max}). Simulations results include the 99% confidence interval computed over 100 simulations.

TABLE I

TOPOLOGY OF FIGURE 5 (A). OPTIMAL AND HEURISTIC RESULTS.

C=1	av. distance	av. # of replica	% not served
Optimal	2.047	1.969	0
Heuristic	2.160 ± 0.162	1.945 ± 0.213	0
C=2	av. distance	av. # of replica	% not served
Optimal	2.362	2.720	0.021
Heuristic	2.352 ± 0.109	2.65 ± 0.176	0.010 ± 0.012

TABLE II

TOPOLOGY OF FIGURE 5 (B). OPTIMAL AND HEURISTIC RESULTS.

C=1	av. distance	av. # of replica	% not served
Optimal	2.435	2.401	0
Heuristic	2.441 ± 0.095	2.537 ± 0.283	0
C=2	av. distance	av. # of replica	% not served
Optimal	-	-	-
Heuristic	2.480 ± 0.119	3.939 ± 0.192	0.038 ± 0.0048

The results show that the heuristic is very close to the optimal behavior. The placement scheme followed by the heuristic only leads to a $2-4\%$ increase in the average users-replica distance and in the number of replicas over what is computed by the optimal strategy. Observe that even for these small experiments, we were not able to compute the optimal solution for $C = 2$ for the larger topology (the number of states was 314928 which multiplied by the number of decision results in about 2 million of variables for the linear problem).

In Table III we report the results for the large topology described in Figure 5 (c), which includes 24 access nodes and 7

TABLE III

TOPOLOGY OF FIGURE 5 (C). HEURISTIC RESULTS.

C=1	av. distance	av. # of replica	% not served
$d_{Max} = \infty$	5.010 ± 0.148	10.438 ± 0.313	0
$d_{Max} = 10$	5.199 ± 0.169	10.414 ± 0.430	0
$d_{Max} = 8$	4.708 ± 0.158	11.279 ± 0.662	0
C=2	av. distance	av. # of replica	% not served
$d_{Max} = \infty$	5.233 ± 0.165	11.180 ± 0.574	0
$d_{Max} = 10$	5.286 ± 0.180	11.188 ± 0.628	$0.06\% \pm 0.05\%$
$d_{Max} = 8$	4.737 ± 0.105	12.770 ± 0.240	$0.07\% \pm 0.17\%$
C=4	av. distance	av. # of replica	% not served
$d_{Max} = \infty$	5.407 ± 0.111	11.252 ± 0.428	0
$d_{Max} = 10$	5.406 ± 0.102	11.522 ± 0.682	$0.06\% \pm 0.07\%$
$d_{Max} = 8$	4.833 ± 0.103	14.637 ± 0.476	$1.36\% \pm 0.79\%$

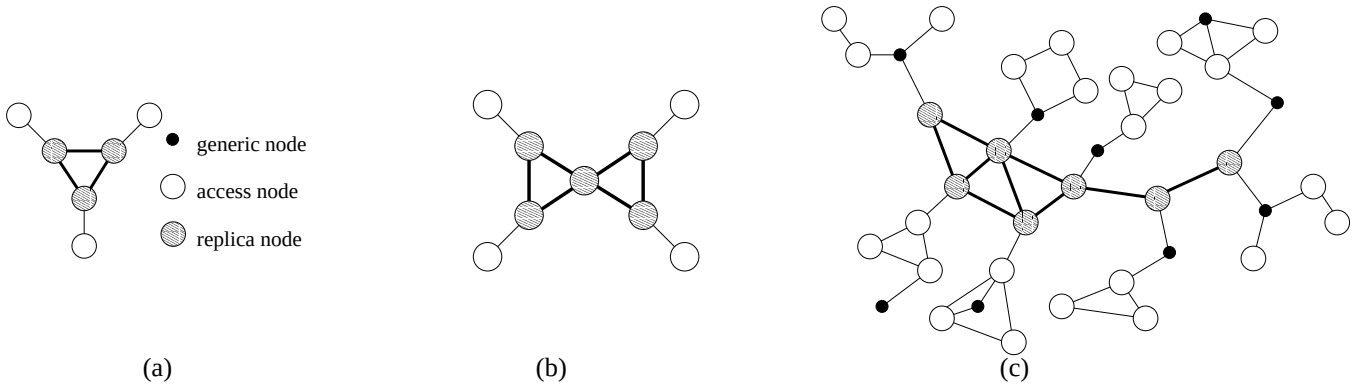

Fig. 5. NETWORK TOPOLOGIES USED IN THE SIMULATIONS.

replica nodes, for $C = 1, 2$ and 4 and $d_{max} = \infty, 10$ and 8. In all these experiments, we set $V_R^{\mathsf{MAX}} = 3$. We observe that, as d_{max} decreases, the average number of used replicas and the probability of not serving user requests increases while the average distance decreases. This reflects the need to place more replicas to meet the stricter constraint on the maximum user-replica distance, and the increased difficulty of satisfying all the requests once the number of possible replica sites that can serve a user is throttled. On the other hand, all quantities increase when the number of different contents hosted in the CDN increases, as more replicas are needed to be able to serve a higher variety of requests and since the limit on the number of replicas per site leads to the additional replicas being placed further and further away from the final user.

Even though we cannot directly compare the behavior of the centralized heuristic with that of the optimal replica placement in this scenario, we can observe that the behavior is close to optimal. Indeed, from the Markov model, we can compute the average number of requests from each access site, which is 0.4, that multiplied by 24 sites yields an average of 19.2 requests overall. Since each replica can serve $K = 2$ requests, we expect for $C = 1$ and $d_{Max} = \infty$ that no less than 10 replicas are necessary to serve - on average - all the requests. We observe that the obtained values are indeed very close to 10.

VI. CONCLUSION

In this paper we have provided a framework for the design of replica allocation schemes dynamically placing and removing replicas in response to changing users demand. By assuming the users requests dynamics to obey to a Markovian model we have first formulated the dynamic replica placement problem as a Markovian decision process. This allowed us to identify an optimal policy for dynamic replica placement that can be used as a benchmark for heuristics evaluation and provides insights on how allocation and deallocation should be performed. Based on the findings obtained through the analytical model we derived and evaluated a centralized heuristic which allocates and deallocates replicas to reflect the requests traffic dynamics, the costs of adding, deleting and maintaining replicas, the servers load and storage limits, and the requirements on the maximum distance of the users from the 'best replica'. The heuristic performance evaluation has shown that the heuristic behavior is very close to that of the optimal placement strategy, and that the heuristic results in good performance in terms of low average number of replicas, low user-replica average distance and low number of requests that cannot be served.

On-going research activity is devoted to the design of distributed heuristics, to the development of a simulation framework for the comparison with the solutions so far proposed in the literature, and to the extension of the performance evaluation to hot spot scenarios.

REFERENCES

[1] `GT-ITM` georgia tech internetwork topology models. `http://www.cc.gatech.edu/fac/ellen.zegura/graphs.html`.
[2] A.T.Bharucha-Raid. *Elements of the theory of Markov processes and their applicatons*. McGraw-Hill, 1960.
[3] Y. Chen, R. Katz, and J. Kubiatowicz. Dynamic replica placement for scalable content delivery. *International Workshop on Peer-to-Peer Systems*, 2002.
[4] D.P.Heyman and M.J.Sobel. *Stochastic Models in Operations Research*. McGraw-Hill, 1984.
[5] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the internet. In *INFOCOM*, pages 31–40, 2001.
[6] M. Karlsson, C. Karamanolis, and M. Mahalingam. A unified framework for evaluating replica placement algorithms. *Technical Report HPL-2002, Hewlett Packard Laboratories*.
[7] J. Keilson. *Markov chain models. Rarity and exponentiality*. Springer-Verlag, New York, 1979.
[8] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, 2001.
[9] M. Rabinovich and A. Aggarwal. RaDaR: a scalable architecture for a global Web hosting service. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1545–1561, 1999.
[10] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed internet replica placement. *Proceedings of WCW'01: Web Caching and Content Distribution Workshop, Boston, MA, June 2001*.
[11] S. Ross. *Applied probability models with optimization applications*. Holden-Day, 1970.