A Framework for Evaluating Replica Placement Algorithms

Magnus Karlsson, Christos Karamanolis, Mallik Mahalingam
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto

HPL-2002-219
August 8th , 2002*

E-mail: karlsson@hpl.hp.com

replica placement algorithms, content delivery networks, evaluation framework, file allocation problem

This paper introduces a framework for evaluating replica placement algorithms (RPA) for content delivery networks (CDN) as well as RPAs from other fields that might be applicable to current or future CDNs. First, the framework classifies and qualitatively compares RPAs using a generic set of primitives that capture problem definitions and heuristics. Second, it provides estimates for the decision times of RPAs using an analytic model. To achieve accuracy, the model takes into account disk accesses and message sizes, in addition to computational complexity and message numbers that have been considered traditionally. Third, it uses the "goodness" of produced placements to compare RPAs even when they have different problem definitions. Based on these evaluations, we identify open issues and potential areas for future research.

A Framework for Evaluating Replica Placement Algorithms

Magnus Karlsson, Christos Karamanolis and Mallik Mahalingam
HP Laboratories
1501 Page Mill Road, Palo Alto, CA 94304, U.S.A.
Contact Author E-mail: karlsson@hpl.hp.com

***Abstract*— This paper introduces a framework for evaluating replica placement algorithms (RPA) for content delivery networks (CDN) as well as RPAs from other fields that might be applicable to current or future CDNs. First, the framework classifies and qualitatively compares RPAs using a generic set of primitives that capture problem definitions and heuristics. Second, it provides estimates for the decision times of RPAs using an analytic model. To achieve accuracy, the model takes into account disk accesses and message sizes, in addition to computational complexity and message numbers that have been considered traditionally. Third, it uses the "goodness" of produced placements to compare RPAs even when they have different problem definitions. Based on these evaluations, we identify open issues and potential areas for future research.**

***Index Terms*— Replica Placement Algorithms, Content Delivery Networks, Evaluation Framework.**

I. Introduction

The design of *replica placement algorithms (RPA)* is one of the foremost problems in Content Delivery Networks (CDN), such as Akamai [1] and Digital Island [2]. These algorithms decide what data to replicate on what storage nodes, in order to achieve improved performance[1] with low infrastructure cost. A number of RPAs have been proposed in the CDN literature, but until now there has been no systematic way to classify and compare them. Moreover, most RPAs are concerned with read-only data and target simple performance metrics. In an earlier paper, we reported that simple caching schemes outperform the best RPAs, if no hard performance guarantees are required [3]. Future CDNs are expected to support modifications, consistency, and various QoS guarantees. Using RPAs is probably the only way to meet such requirements. Thus, RPAs for next-generation CDNs is an issue that needs to be investigated.

[1]Performance could be, for example, latency, throughput or availability.

The replica placement problem has also been studied extensively in several other distributed computing fields, including the file assignment problem [4], distributed databases [5] and data management [6], to mention a few. These fields require RPAs that may consider writes, consistency, availability, strict and compounded guarantees, update propagation bounds, etc. Several questions are raised as a consequence: how are all these approaches related to each other; how can one systematically represent and evaluate algorithms for use in CDNs; what RPAs from other fields can CDNs use; which areas provide opportunities for future research.

In this paper, we propose a framework for comparing and evaluating replica placement algorithms for CDNs. Our framework defines an RPA as a *problem definition* and a *heuristic*. The problem definition consists of a cost function to be minimized under some constraints. The heuristic is used to produce near-optimal solutions for the resulting problem, which is usually NP-complete.

Section II introduces a set of primitives to describe RPAs from multiple disciplines. Using these primitives serves three purposes. First, it clarifies the fundamental differences between existing algorithms. Second, it simplifies comparing new algorithms with the prior art. Third, given a problem definition, it identifies heuristics that have been used for that problem.

We find that many potential future problems in CDNs have solutions in other fields. However, section III shows that existing algorithms do not support the scales of CDNs. We propose an analytic approach for estimating the *decision time* of algorithms. Traditionally, only computation and message complexities have been considered. We show that message sizes and disk access times due to memory constraints also need to be taken into account to provide accurate estimates. RPAs that could be of use in future CDNs, usually scale even worse than today's CDN RPAs. This highlights the need for research in scalable algorithms that solve more complex replica placement problems in CDNs.

Section IV compares algorithms based on how "good" the produced placement is. A problem with the existing CDN literature is that algorithms are compared using minimized cost function values. Thus, comparison of algorithms with different problem definitions is impossible even when they target the same goal and system. One way to circumvent this problem is to compare their respective impact on the system performance or cost. In particular, we developed *Coeus*, an RPA generator that can produce the placement of most RPAs described in our framework to facilitate comparisons against the prior art. The power of this method is illustrated by comparing a number of algorithms never compared before, as they have diverse problem definitions. Finally, we present related work in Section V and conclude in Section VI.

II. Replica Placement Algorithms

Replica placement is typically formulated as a *problem definition* that approximates the overall goal (performance or cost improvement), the workload and the target system. These problems are usually NP-complete. Thus, they usually require *heuristics* to find approximate solutions within feasible time. In this paper, by *algorithm* we refer to a heuristic applied to a specific problem definition.

To limit the scope of this paper, we do not consider algorithms that aim solely at improving the availability or reliability of the system, nor algorithms dealing only with migration. We only consider RPAs that are applicable to general or tree topologies, as we are interested in Internet-like topologies.

A. System Model and Goal

The system considered in this paper is a data repository consisting of a set of *nodes* interconnected with *links*. Nodes store *objects* that represent data, such as files, websites or volumes. *Clients* connect to the nodes to access the objects stored there.

The goal of the replica placement problem is to decide the location of object replicas in the system, in order to either *maximize the client perceived performance given an existing infrastructure*, or *minimize the infrastructure's cost given a specified system performance*. This system goal is abstracted into a problem definition that is used as an optimization goal.

B. Problem Definition Framework

The replica placement problem can be formally stated as follows. The system consists of a set of clients C, nodes N, objects K, and links L. If there is a physical communication channel between two nodes, a link is added between these nodes. Each client $i \in C$ is assigned to a node $j \in N$ for each object $k \in K$, incurring a specific cost according to a *cost function*. For example, such a function may reflect the average latency for clients accessing objects in the system's nodes. An extensive sample of cost functions is shown in Table I.

This problem formulation is augmented with a number of *constraints*. The binary variable y_{ijk} indicates whether client i sends its requests for object k to node j; x_{jk} indicates whether node j stores object k. The following four constraints are present in most problem definitions (the numbers refer to the equations below): (1) states that each client can only send requests for an object to exactly one node; (2) states that only nodes that store the object can respond to requests for it; (3) and (4) imply that objects and requests cannot be split. Optional additional constraints are described later in this section. The basic problem is thus to find a solution of either minimum or maximum cost that satisfies constraints (1) – (4).

$$\sum_{j \in N} y_{ijk} = 1 \quad \forall i, k \tag{1}$$

$$y_{ijk} \leq x_{jk} \quad \forall i, j, k \tag{2}$$

$$x_{jk} \in \{0, 1\} \quad \forall j, k \tag{3}$$

$$y_{ijk} \in \{0, 1\} \quad \forall i, j, k \tag{4}$$

The cost functions of existing RPAs, as shown in Table I, use the following parameters:

Reads $(reads_{ik})$: The rate of read accesses by a client i to an object k. This might also be reflected as the probability of an access to an object $(P(reads_{ik})_t)$ within t time units.

Writes $(writes_{ik})$: The rate of write accesses by a client i to an object k.

Distance $(dist_{ij})$: The distance between a client i and a node j, represented with a metric such as network latency or link "cost". For update propagation costs, some algorithms use the minimum spanning tree distance between a node j and all the other nodes with a copy of object k, denoted mst_{jk}.

Storage Cost (sc_{jk}) : The cost of storing object k at node j. The storage cost might reflect the size of the object, the throughput of the node, or the fact that a copy of the object is residing at a specific node, also called *replication cost*.

Object Size $(size_k)$: The size of object k in bytes.

Access Time $(acctime_{jk})$: A time-stamp indicating the last time object k was accessed at node j.

Hit Ratio (hr_{ij}) : Hit ratio of any cache on the path from i to j.

In the literature, a number of additional constraint primitives might be added to constraints (1) – (4) of the problem definition:

Storage Capacity (SC) :
$\sum_{k \in K} size_k \cdot x_{jk} \leq SC_j, \ \forall j.$ An upper bound on the storage capacity of a node.

Load Capacity (LC) :
$\sum_{i \in C} \sum_{k \in K} (reads_{ik} + writes_{ik}) \cdot y_{ijk} \ \leq \ LC_j$, $\forall j.$ An upper bound on the load, characterized as the rate of requests a node can serve.

Node Bandwidth Capacity (BW) :
$\sum_{i \in C} \sum_{k \in K} (reads_{ik} + writes_{ik}) \cdot size_k \cdot y_{ijk} \leq BW_j, \ \forall j.$ A constraint on the maximum rate of bytes a node can transmit.

Link Capacity (CL) :
$\sum_{i \in C} \sum_{k \in K} (reads_{ik} + writes_{ik}) \cdot size_k \cdot y'_{lik} \leq CL_l, \ \forall l.$ A bandwidth constraint on the link between two nodes. $y'_{lik} = 1$, if client i uses link l to access object k, otherwise it is zero. Some problem definitions separate reads, writes and object replications. The notations for those are y^r_{ilk} for read accesses, y^w_{ilk} for writes, and y^m_{ilk} for objects being replicated.

Number of Replicas (P) : $\sum_{j \in N} x_{jk} \leq P, \ \forall k.$ A constraint limiting the number of replicas placed.

Origin copy (OC) : $x_{jk} = 1$ for given j and k. Specifies the location of the original copy of an object.

Delay (D) : Specifies the desired maximum response time for requests in the system. Its mathematical definition is found in [7].

Availability (AV) : Specifies the desired minimum availability of objects in the system. Its mathematical definitions are found in [7], [8].

Table I maps replica placement problem definitions from many disparate fields into the proposed cost-function primitives and constraints. The table lists only problem definitions that we believe might be useful in a CDN. A more comprehensive list can be found in [40]. The problem definitions have been broken down into three main groups. The Group 1 ignores client accesses; the Group 2 only accounts for read accesses; the Group 3 considers both read and write accesses, including consistency requirements. These groups are further divided into four categories according to whether a problem definition takes into account *single* or *multiple objects*, and whether it considers *storage costs*. Single-object formulations cannot handle intra-object constraints, such as storage constraints, but they are easier to solve.

The drawback of the problem definitions in Group 1 is that they place the P replicas of every object, in the same P nodes. Clearly, this is not practical, when many objects are placed in the system. However, they are useful as a substitute of Group 2 problem definitions, if the objects are accessed uniformly by all the clients in the system and the utilization of all nodes in the system is not a requirement. In this case, Group 1 algorithms can be orders of magnitude faster than the ones for Group 2, because the placement is decided once and it applies to all objects.

Almost all problem definitions proposed in the literature for use in CDNs fall under Group 2. They are applicable to read-only and read-mostly workloads. Problem definitions (3), (5), (6), (8) and (9) have all been used in CDNs. The two main differences between them are whether they consider single or multiple objects, and whether they consider storage costs or not. The cost function in (7) also captures the impact of allocating large objects and could possibly be used when the object size is highly variable. (4) has been proposed for allocating caches. Its distance parameter consists of the distance between the client and the cache, plus the distance between the cache and the node for all cache misses. In a CDN problem definition, the distance is measured between the client and the closest node that has a copy of the object.

In the context of CDNs, storage costs (sc_{jk}) could be used in order to minimize the amount of changes to the previous placement. As far as we know, there has been no evaluation in a CDN, of the benefits of taking this into consideration. Another open question is whether storage, load, nodal bandwidth and link capacity constraints need to be considered. If so, are there any scalable good heuristics for such problem definitions? There is also little research in CDN algorithms that enforce QoS guarantees, such as client perceived latency.

Considering the impact of writes, in addition to that of reads, is important, if content providers and applications are able to modify documents in a CDN. This is the main characteristic of Group 3, which contains problem definitions that could be of interest to future CDNs. These problem definitions represent the consistency protocol in many different ways. For most of them, the cost is the number of writes times the distance between the client and the closest node that has the object, plus the cost of distributing these updates to the other replicas of the object. In (10), (12), (17), (18) and (19), the updates are distributed in the system using a minimum spanning tree. In (11) and (14), one update message is sent from the writer to each other copy. In (15) and (16), it is not specified how updates are propagated. (18) and (21) also consider the cost of exchanging information and (20) considers channel cost.

The other main difference among the above definitions is that (12), (17), (18) and (21) minimize the maximum link congestion, while the rest minimize the average client

TABLE I

THE PROBLEM DEFINITIONS USED BY ALL THE HEURISTICS DEALT WITH IN THIS PAPER. THE CONSTRAINTS COLUMN SHOWS THE CONSTRAINTS THAT HAVE BEEN USED IN THE LITERATURE IN CONJUNCTION WITH A SPECIFIC COST FUNCTION, NOT ALL POSSIBLE ONES. THE VARIOUS COMPONENTS OF THE COST FUNCTION MIGHT BE WEIGHTED. HOWEVER, THESE ARE NOT SHOWN IN THE TABLE.

#	COST FUNCTION	CONSTRAINTS	REFERENCES
Group 1: Does not consider any object accesses.			
	Single Object		
(1)	$\max_{i\in C, j\in N} dist_{ij} \cdot y_{ijk}$	P	[9], [10], [11]
(2)	$\sum_{i\in C}\sum_{j\in N} dist_{ij} \cdot y_{ijk}$	P	[9], [11]
Group 2: Considers only read accesses to objects.			
	Single Object		
(3)	$\sum_{i\in C}\sum_{j\in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk}$	P, LC, BW	[9], [11], [12], [13], [14], [15]
(4)	$\sum_{i\in C}\sum_{j\in N} reads_{ik} \cdot (hr_{ij} \cdot dist_{ij} + (1 - hr_{ij}) \cdot (dist_{ij} + dist_{js}))$	P	[16], [17]
	Multiple Objects		
(5)	$\sum_{i\in C}\sum_{j\in N}\sum_{k\in K} reads_{ik} \cdot dist_{ij} \cdot y_{ijk}$	SC, OC	[18], [19]
(6)	$\sum_{i\in C}\sum_{j\in N}\sum_{k\in K} P(reads_{ik})_t \cdot dist_{ij} \cdot y_{ijk}$	SC, BW	[20], [21]
(7)	$\sum_{i\in C}\sum_{j\in N}\sum_{k\in K} reads_{ik} \cdot dist_{ij} \cdot size_k \cdot y_{ijk}$	SC	[22]
	Single Object + Storage Cost		
(8)	$\sum_{i\in C}\sum_{j\in N} (sc_{jk} \cdot x_{jk} + dist_{ij} \cdot reads_{ik} \cdot y_{ijk})$	LC, BW	[13], [23], [24], [25]
	Multiple Objects + Storage Cost		
(9)	$\sum_{i\in C}\sum_{j\in N}\sum_{k\in K} (sc_{jk} \cdot x_{jk} + dist_{ij} \cdot reads_{ik} \cdot y_{ijk})$	-	[24]
Group 3: Considers both read and write accesses with update propagation.			
	Single Object		
(10)	$\sum_{i\in C}\sum_{j\in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot (dist_{ij} + mst_{jk}) \cdot y_{ijk}$	P	[26], [27], [28]
(11)	$\sum_{i\in C}\sum_{j\in N} (reads_{ik} \cdot dist_{ij} + writes_{ik} \cdot (dist_{ij} + \sum_{n\in N, n\neq j} dist_{jn} \cdot x_{nk})) \cdot y_{ijk}$	P	[29]
(12)	$\max_{l\in L} \sum_{i\in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk})/CL_l$	-	[6]
	Single Object + Storage Cost		
(13)	$\sum_{i\in C}\sum_{j\in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot (dist_{ij} + mst_{jk}) \cdot y_{ijk} + \sum_{j\in N} sc_{jk} \cdot x_{jk}$	P	[30], [31], [32]
(14)	$\sum_{i\in C}\sum_{j\in N} (reads_{ik} \cdot dist_{ij} + writes_{ik} \cdot (dist_{ij} + \sum_{n\in N, n\neq j} dist_{jn} \cdot x_{nk})) \cdot y_{ijk} + \sum_{j\in N} sc_{jk} \cdot x_{jk}$	P	[33]
(15)	$\sum_{j\in N} (\sum_{i\in C, i\neq j} reads_{ik} \cdot dist_{ij} \cdot y_{ijk}) + (sc_{jk} + update\_cost_{jk}) \cdot x_{jk}$	-	[34]
(16)	$\sum_{i\in C}\sum_{j\in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot update\_dist_{ij} \cdot availability_{ijk} \cdot x_{jk} + \sum_{j\in N} sc_{jk} \cdot x_{jk}$	AV	[8]
(17)	$\max_{l\in L} (\sum_{i\in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) + \sum_{j\in N} sc_{jk} \cdot y^m_{jlk})/CL_l$	-	[32]
(18)	$\max_{l\in L} (\sum_{i\in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) + \sum_{j\in N} sc_{jk} \cdot y^m_{jlk} + info\_exchange_{lk})/CL_l$	-	[35]
	Multiple Objects + Storage Cost		
(19)	$\sum_{j\in N}\sum_{k\in K}\sum_{i\in C} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot (dist_{ij} + mst_{jk}) \cdot y_{ijk} + \sum_{j\in N}\sum_{k\in K} sc_{jk} \cdot x_{jk}$	SC	[36], [37], [38]
(20)	$\sum_{i\in C}\sum_{l\in L}\sum_{k\in K} channel\_cost_l \cdot y'_{ilk} + \sum_{j\in N}\sum_{k\in K} sc_{jk} \cdot size_k \cdot x_{jk}$	CL, D, AV	[7]
(21)	$\max_{l\in L} (\sum_{k\in K}\sum_{i\in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) + \sum_{j\in N} sc_{jk} \cdot y^m_{jlk} + info\_exchange_{lk})/CL_l$	SC	[39]

access latency or other client-perceived costs. Minimizing the link congestion would be useful, if bandwidth is scarce in the CDN.

Note, that none of these problem definitions considers load or nodal bandwidth constraints and only (19) and (21) consider storage constraints. If these constraints are shown to be important, then there are open research issues in this space. Another interesting question is how these problem definitions can be extended with constraints that bound the update propagation time, an important property of a CDN. Some of the problem definitions consider client perceived latency and availability guarantees. However, as we will see later in the paper, the corresponding heuristics proposed in the literature are not scalable. Thus, there are more research opportunities in designing scalable algorithms.

C. Heuristic Primitives

The NP-complete problem defined in the previous section is solved using heuristics.[2] We have found that heuristics can be characterized along three axes: *metric scope*, *approximation method* and *cost function simplification*.

Metric Scope. It refers to the clients, nodes, objects and links that are considered when placing objects. The chosen scope affects the techniques used. For example, if only one node is considered, the heuristic is decentralized and it is independently executed on every single node in the system. On the other hand, if all nodes are considered, it is executed in a centralized way. If only local object knowledge is specified, then only the objects stored in the local node or objects that have been referenced by the local node are considered.

Approximation Method. It is the technique used to make the placement decisions. The methods used in the heuristics are:

Ranking ($R(plain/greedy)$). Compute the cost impact of all possible combinations (within the metric scope) of placing one extra object on one node. Sort these costs and select the best one that does not violate any constraints. If a constraint is violated, try the next placement in the list. A *greedy* ranking heuristic [41] recomputes the cost function after each object is placed. Ranking is a generic approximation method and can be used for all problem definitions.

Fixed Threshold ($T(threshold)$). An object is placed at a specific node, if the cost function is above or below a specified threshold. This approximation method is usually independent of the problem definition.

Improvement ($I(method)$). These methods try to improve on a solution by introducing small changes. The m*-distance* improvement heuristic [42] evaluates permutations of the solution where at most m bits in the decision variables are changed. There are numerous other interchange methods, many of them referenced in [42]. The improvement methods are independent of the problem definition.

Relaxation Techniques ($Re(method)$). Encompass a number of techniques. *Lagrangian relaxation* [43] relaxes the constraints of the problem definition by moving them into the cost function. *Linear relaxation* [43] relaxes the integer constraints of the original problem. It is thus possible to use fast linear optimization techniques, instead of slow integer ones. These algorithms have to be designed specifically for a problem definition.

Dynamic Programming (DP). This method saves intermediate results to avoid recalculation [41]. Rules can be applied in order to merge intermediate results. This method is specific to the problem definition.

Parametric Pruning (PP). It is a technique specific to the problem definition, that prunes irrelevant portions of the search space based on a conservative estimate of the optimal cost [43].

Hierarchical (H). All the aforementioned approximation methods can be used in a hierarchical fashion. For example, ranking can be used on each node at the leaf level. The results from the ranking are then aggregated by the nodes on the next higher level that perform another ranking using this aggregated data, and so on up to the root node [21].

Multi-phase ($M(method_1, method_2, ...)$). Some of the techniques described above can be combined in certain ways. For example, greedy ranking can be followed by an improvement heuristic [11], [21]. Multi-phase techniques can also be combined with hierarchal methods [21].

Cost Function Simplification. It is made to the original cost function of the problem definition. For example, the problem definition might specify the cost function $\sum_{i \in C} \sum_{j \in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk}$. But for example, the Hotspot heuristic (see Table II) disregards the distance and uses just $\sum_{i \in C} \sum_{j \in N} reads_{ik} \cdot y_{ijk}$. The simplified cost function might work well anyway.

Table II lists previously proposed heuristics for replica placement problems, mapped into our framework. The table serves two main purposes. The first is to qualitatively compare existing heuristics from the literature. The main observation is that problem definitions from Groups 1 and 2 have been used in combination with generic heuristics, while this is not the case with Group 3 (with the exception of problem definition (15)). Another observation is that many of the proposed ranking heuristics are quite similar or even the same. The second purpose of the table is that new heuristics can be crisply described and reported. Note that the table and the primitives only give a high-level view of the heuristics; in order to faithfully implement one of them, the source reference has to be consulted. A plethora of caching algorithms other than basic LRU are not included due to space constraints. The excluded caching algorithms differs in their cost function simplifications and their metric scopes if cooperative caching is used.

Some heuristics in Table II have two approximation

[2]For briefness in this paper, heuristic refers to traditional heuristics as well as approximation algorithms.

TABLE II

EXISTING HEURISTICS MAPPED INTO OUR FRAMEWORK. A "?" MEANS THAT IT IS NOT CLEAR FROM THE PAPER WHAT THIS ENTRY IS. "CHILD" MEANS THAT THE METRIC SCOPE IS THE CHILDREN OF A NODE IN THE TOPOLOGICAL TREE. "VICINITY" MEANS A NUMBER OF ENTITIES CLOSE TO A NODE (SEE THE CORRESPONDING PAPER FOR DETAILS). THE PROBLEM DEFINITION COLUMN LISTS THE PROBLEM DEFINITIONS THAT A HEURISTIC HAS BEEN APPLIED TO IN THE LITERATURE.

		Metric Scope			**Cost Function**	**Problem**
Heuristic	**Approximation Method**	**Client**	**Node**	**Object**	**Simplification**	**Definition**
Ranking heuristics						
Greedy Global [18], [14]	R(greedy)	all	all	all	-	3,5
Ranking Local [18], [20]	R(plain)	local	one	local	-	5,6
Popularity [18]	R(plain)	local	one	local	$reads_{ik}$	5
Hotspot [14]	R(plain)	vicinity	all	all	$reads_{ik}$	3
Fan-Out Based [11], [15]	R(plain)	indep	all	indep	$fanout_j$	1,2,3
Ranking Dist [20]	R(plain)	local	one	local	$dist_{ij}$	6
LRU Caching	R(plain)	local	one	local	$acctime_{jk}$	Group 2
Improvement Heuristics and Combinations						
2-distance [34], [13]	I(2-dist)	all	all	all	-	3,8,15
l-Greedy [11]	M(R(greedy),I(m-dist))	all	all	all	-	1,2
Hierarchical [19], [21]	H M(R(plain),I(2-dist))	child	child	child	$dist_{ij} = \max_{i,j} dist_{ij}$	5,6
Threshold Heuristics						
RaDaR [44], [45]	Place: T($reads_{ik} / \sum_{i \in C} y_{ijk} > C$) Delete: T($\sum_{i \in C} y_{ijk} < 1$)	local local	two one	local local	? ?	? ?
Awerbuch FAP [36], [38]	Reads: T($reads_{ik} > C$) Writes: R(plain)	all all	all all	all all	$reads_{ik}$ $\sum_{i \in Writers} dist_{ij}$	19 19
Dist Awerbuch [36], [38]	Reads: T($reads_{ik} > C$) Writes: R(plain)	local local	vicinity writers	local local	$reads_{ik}$ $\sum_{i \in Writers} dist_{ij}$	19 19
Edge Strategy [35]	Reads: T($reads_{ik} \geq C$) Writes: T($writes_{ik} \geq C$)	local local	two two	local local	- -	18 18
Other Heuristics (all problem definition specific)						
Min k-Center [10], [11]	PP	all	all	indep	$dist_{ij}$	1
Tree-based [24], [30], [16], [12], [14], [26]	DP	all	all	all	-	3,4,8,9, 10,11,13
Linear [22]	RE(linear)	all	all	all	-	3,5,7,8,9
Linear+Imp [8]	M(RE(linear),I(drop),I(add/drop))	all	all	all	-	16
Lagrangian [33]	RE(lagrangian)	all	all	all	-	3,5,14
Lagrangian+Imp [7]	M(RE(lagrangian),I(add/drop))	all	all	all	-	20

methods, because they were designed to be on-line algorithms. The classical definition of an *on-line* algorithm is that the guarantees that come with it are valid in a setting where future accesses are not known. Conversely, an *off-line* algorithm's guarantees are only valid for settings where the future is known. Most on-line algorithms are designed to be invoked as frequently as after each single access. Thus, they cannot generally afford to re-evaluate the whole placement at every single invocation. A fast execution time can be accomplished, for example, by having one heuristic for placing an object and another one for deleting an object. For the algorithms that do not have an explicit delete heuristic (typically, off-line algorithms), object deletion is implicit—the heuristic decides about object placement and nodes that are not included in the placement, delete the object if they have it. In the evaluation in Section IV, we consider all algorithms in the on-line setting, as it is more realistic, even though any original off-line guarantees are lost.

III. DECISION TIME

Decision time is the time required for an algorithm to make a placement decision for all the objects in a specific system. When the algorithm is distributed, this is the maximum completion time among all nodes in the system. Previously, only computation time and the time due to the number of messages sent have been considered, when estimating decision times. In this section, we advocate that message sizes and disk accesses also have to be taken into account to provide a good estimate. We develop an analytic model that, using the primitives of the previous section, estimates the decision time of many algorithms according to the following formula:

$$\begin{aligned} T_{tot} = \quad & C_{comp} \cdot T_{comp} + & (Comp) \\ & C_{msg} \cdot T_{msg} + (S_{tot} - S_{local}) \cdot T_{msgpb} + & (Msg) \\ & \max(S_{req} - M, 0) \cdot C_{disk} \cdot T_{diskpb} & (Disk) \end{aligned}$$

The first component of the total decision time is the computation time (*Comp*). It is the computational complexity of the algorithm times the average time it takes to perform one iteration or calculation ($C_{comp} \cdot T_{comp}$), when the data are in the main memory. The message cost (*Msg*) is the time spent on sending and receiving messages. It does not include message transfer times, as we assume asynchronous message passing. That is, useful computations can be scheduled while messages are in transit. The message cost consists of two components: one that is proportional to the number of messages ($C_{msg} \cdot T_{msg}$, where T_{msg} is the time for sending and receiving a zero-length message), and one that is proportional to the size of the transmitted data. The latter is the total size of the data needed to make a decision minus the local data-set on the node ($S_{tot} - S_{local}$) times T_{msgpb}, the time spent per byte of transmitted data. The disk cost (*Disk*) is the time required for accessing data on the disk, when the data do not fit in the node's main memory. It is the amount of memory required to make the decision minus the effective amount of memory in the node ($S_{req} - M$). This is multiplied by the number of times this data are reused before the decision is made (C_{disk}) and the time it takes to read one byte of data from the disk (T_{diskpb}).

We use this analytic model to estimate the decision times of RPAs without having to build and test the actual algorithms in a real or emulated environment. This approach is particularly important when the target systems are large and when new algorithms are designed and need to be evaluated quickly.

A. Accuracy of the Analytic Model

In this section, we compare the decision times estimated using the proposed analytic model with the times of actual algorithm executions, for systems of relatively small size. The model provides sufficiently accurate estimates for approximate comparisons. Moreover, we demonstrate that computation, message and disk costs have all to be taken into account in order to provide a good estimate.

We have developed a tool called `Coeus`, that generates and evaluates RPAs. It implements most of the problem definitions and heuristics described in the previous sections. Using `Coeus`, we can execute RPAs for different system sizes, and measure the actual decision times of those algorithms.

Table III shows the values of the various parameters of the analytic model. The parameters in the top two rows are measured by means of actual RPA implementations in `Coeus`. The presented values were obtained on a workstation with a 552MHz PA8600 processor and 512MB of memory. The entire data set was kept in memory for the T_{comp} measurements. For T_{diskpb}, the data set was larger than the available memory. T_{msg} and T_{msgpb} were chosen so that a node receives 10^3 10KB messages per second, a value typical for static web serving. In the case of Ranking (R) and Improvement (I) approximation methods, these values are applicable to all possible heuristics. In the case of Lagrangian relaxation, the values are applicable only to the specific problem definition used; new measurements are required for other problem definitions.

The other parameters are calculated analytically. The computational complexities (C_{comp}) in Table III are deduced by studying the C-code of the implementations or by getting the computational complexities from the original papers. I_{2dist} and I_{lagr} in the equations are the number of iterations that the two approximation methods execute. S_{tot} is calculated by adding up the number of variables the node requires to make a decision, assuming 8 byte variables. S_{local} is calculated in the same way, but only for data gathered at the local node.

The accuracy of the analytic model in predicting the decision times is evaluated for six heuristics applied to problem definition (3). We compared the decision times estimated by the analytic model using the parameter values of Table III, versus the actual decision times observed for the actual execution of the algorithms in `Coeus`. For the comparisons, we used systems of 10 different sizes: from 1,000 up to 10,000 (in increments of 1,000) clients and nodes. Problem definition (3) considers only a single object at a time. The comparisons show that the model error is under 34% for our test cases, as shown in Figure 1. The larger errors occur for smaller systems, mainly because the data start to fit into the processor caches. We have also validated heuristics for problem definitions (1), (2), (5), (7), (8) and (9) showing errors of similar scale as shown for (3).

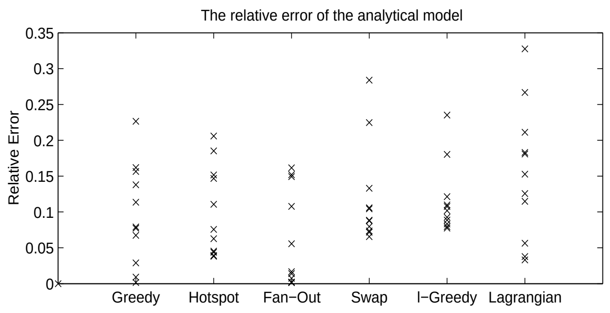


Fig. 1. The relative error ($|T_{real} - T_{pred}|/T_{real}$) of the analytic model for six heuristics and various systems.

TABLE III

THE PARAMETERS FOR THE DECISION TIME MODEL. C, N, AND K REPRESENT THE METRIC SCOPE OF CLIENTS, NODES AND OBJECTS, RESPECTIVELY. R IS THE NUMBER OF REPLICAS PLACED BY THE NODE THAT MAKES THE DECISION EACH TIME IT RUNS THE ALGORITHM. u_{dist} AND u_{dem} ARE 1 IF THE HEURISTIC USES THE DISTANCE AND DEMAND MATRIX, RESPECTIVELY.

Parameter	Value	Parameter	Value	Parameter	Value
T_{msg}	10^{-4} s	T_{msgpb}	10^{-7} s	T_{diskpb}	10^{-7} s
T_{comp} R()	10^{-7} s	T_{comp} I(2-dist)	10^{-8} s	T_{comp} Re(Lagr)	$1.5 \cdot 10^{-7}$ s
M	256 MB	C_{msg}	$N - 1$	C_{comp} R(greedy)	$CNRK$
C_{comp} R(plain)	CNK	C_{comp} I(2-dist)	$I_{2dist}CNK$	C_{comp} Re(Lagr)	$I_{lagr}CNRK$
S_{req} R(greedy)	$8(NCu_{dist} + Cu_{dem})$	S_{req} R(plain)	$8(NCu_{dist} + Cu_{dem})$	S_{req} I(2-dist)	$8(NCu_{dist} + Cu_{dem})$
S_{req} Re(Lagr)	$8(NC + 2C + 4N)$	C_{disk} R(greedy)	RK	C_{disk} R(plain)	K
C_{disk} I(2-dist)	$K\sqrt{I_{2dist}}$	C_{disk} Re(Lagr)	KI_{lagr}		

Our model can be used to provide sufficiently precise decision time estimates, in two cases. First, when a new algorithm is proposed that uses a problem-definition-independent approximation method, its decision time can be estimated using the model with pre-computed parameter values from a table such as Table III. Thus, these algorithms do not have to be implemented in order to evaluate the decision time. Second, when a heuristic with a problem-definition-specific approximation method is proposed, it will have to be implemented first in a tool such as `Coeus`. Using small-scale executions, the parameter values of the model can be measured and then the model can be used to estimate the decision times for large-scale (and therefore infeasible to emulate) deployments.

B. Decision Time of Existing Heuristics

This section discusses the decision times of some algorithms that have been proposed in the literature, using the analytic model introduced in the previous section. Figure 2 depicts the decision times for six representative heuristics from Table II optimizing problem definition (3) with a number of replica constraint. The numbers of nodes and clients are 10^3 and 10^4 respectively for the two graphs, and the number of objects is 10^4. There are three main conclusions to be drawn. First, some heuristics are dominated by computation cost, some by one of the two message costs, and others by disk cost, signifying the importance to take all four factors into account. For example, someone looking only at the computation time for the greedy heuristic, in a system with 10^4 nodes, might conclude that it is feasible, while it will in fact take several weeks longer due to disk accesses on our system. With our model, disk accesses are correctly estimated to be the dominant decision time factor. The dominant factor for a heuristic may change for different system sizes and problem definitions (see, for example, the cost of *greedy* in the two graphs). Second, the difference between the decision times of various heuristics is very large (orders of magnitude). Third, even at the relatively small scale studied here, some heuristics are infeasible as they take weeks to terminate.

Figure 3 shows the decision costs of a system with 10^6 clients and nodes and 10^8 objects, a moderately-sized global storage repository such as OceanStore [46]. Due to the system's size, we cannot measure the I variables as in Figure 2. Instead, we assume that $I_{2dist} = 10^8$ is enough when seeded with a placement from some other heuristic. We believe that using Swap on its own at such scales is not meaningful. In the case of the Lagrangian relaxation algorithm, we assume that it terminates after 300 iterations.

For such scales, not a single existing heuristic is feasible. Fan-Out has a feasible decision time, but as it places all objects in the same nodes, it is not practical when 10^8 objects are considered. There are two main observations to take away from this figure. First, only a heuristic with a low metric scope can be feasible at these scales. Second, in order to be able to solve the replica placement problem for systems of this size, there is a need to aggregate data (clients, nodes, objects, and links) down to a point where the scale of the problem becomes manageable with low metric scope heuristics.

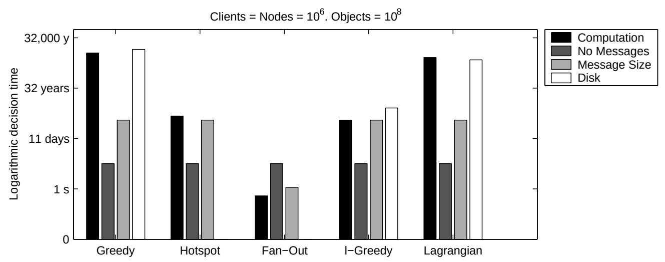


Fig. 3. The decision costs for five heuristics at OceanStore scales.

IV. Goodness of Placement

In the literature, the goodness of a heuristic in solving a specific problem definition is evaluated using, for example, competitive ratios or minimized cost function values.

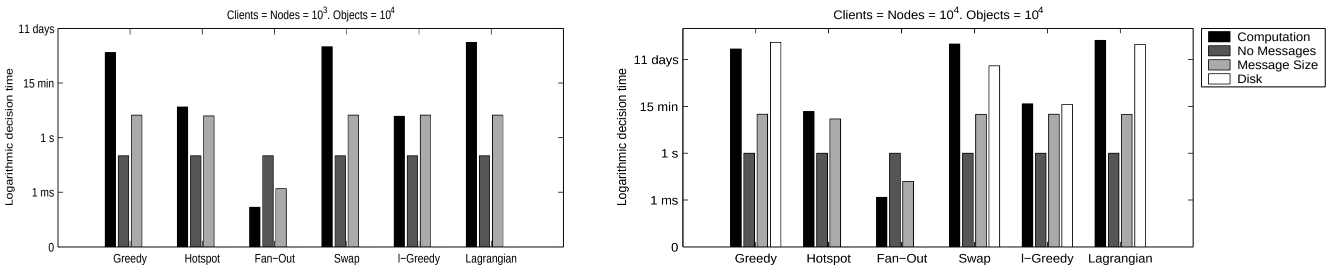


Fig. 2. The decision costs for some existing heuristics at medium scales. At the scale of the left diagram, there is no disk access time because all the data fits in memory. $I_{2dist} = 10^4$ and $I_{lagr} = 115$.

However, in CDNs (and many other fields), algorithms have been proposed that use different problem definitions for the same system and goal. Thus, the goodness of the placements of these algorithms cannot be compared using the aforementioned methods, even though they should be compared as they target the same system and goal. To tackle this problem, we propose to quantitatively evaluate an algorithm's impact on the performance or cost of the system it is intended to be used in. In other words, we evaluate the achieved system performance or cost due to the placement produced by various RPAs, and not how far from the optimal cost function value a solution is. This is useful even in the context of a single problem definition as shown in [47], as it reflects how well the problem definition corresponds to the system and goal. We will focus on the evaluation of system performance, as the evaluation of money cost is straight-forward.

When comparing the placement produced by algorithms with different problem definitions, it is important to factor in the penalty of not meeting certain constraints or taking certain parameters into account. When the performance impact of a placement is measured on an actual system, such factors are automatically reflected on the result. However, when the performance is evaluated using analytic models or simulators, the constraint violations and parameter exclusions with their associated costs have to be explicitly included in the model.

The proposed goodness evaluation approach is illustrated by comparing a number of RPAs for use in CDNs, that have never been compared before. This is not intended to be an exhaustive comparison of algorithms, just an example of the approach. We base our evaluation on system simulation.

A. *Experimental Methodology*

Three ingredients are required to compare the goodness of the performance-based placement of different RPAs: a performance metric, a representative workload, and the produced placements. The *placements* of the RPAs are produced by `Coeus`, described in Section III-B. The other two issues are discussed in the following paragraphs.

The performance metric to be used depends on what the system is used for. For a CDN, we use a client perceived latency threshold as the performance metric. The evaluation refers to the Cumulative Distribution Function (CDF) of the latency to access objects, given the produced placements. The larger the ratio of access under the threshold, the better.

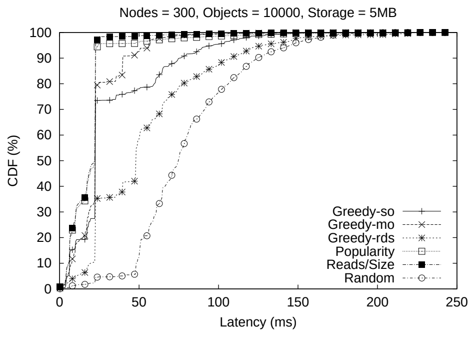
We use the World Cup 1998 web logs [48] to generate a realistic workload for a CDN. To reduce the client population to a tractable size for our evaluations, we used the logs of days 50 to 59 and clustered all encountered client IP addresses according to the Autonomous System (AS) they belong to.[3] This clustering preserves the topological locality of clients and reduces the number of clients from 2.6M to 5.4K. These clusters represent both the clients in the system and the nodes on which objects may be placed. Matrix $reads_{ik}$ is obtained by counting the number of requests each such cluster generates. To simulate systems with less than 5.4K nodes, we choose the desired number of nodes in a way that preserves the original access distribution from [48]. Each URL is treated as a separate object. There are 38K objects in the 10 days of the WorldCup98 log that we studied, which are reduced to 10K objects by random selection.

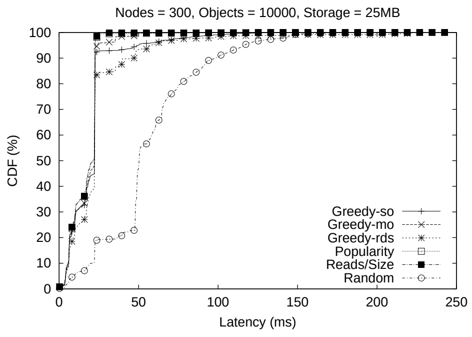
In an ideal world, matrix $dist_{ij}$ would represent the average latency between nodes i and j. However, this is impossible to measure, unless one happens to be on the specific routing paths. So, we obtain latency approximations by counting the number of AS-level hops between two nodes. It has been shown that this is a fair approximation of actual latencies on the Internet [50]. To turn these hop numbers into latencies with some variation (as the actual latencies undoubtedly have), we use the formula $Latency = \lfloor 50 \cdot (hops + \Delta) \rfloor\ ms$, where Δ is a random value in $[-0.5.. + 0.5]$.

B. *Evaluation Example*

To illustrate our systems-centric approach in evaluating RPAs, we address two questions. First, if there is, for our

[3]This clustering is similar to that of [49], but results in fewer and therefore coarser clusters.





	Storage Capacity	
Algorithm	5MB	25MB
Greedy-so	18.3	12.0
Greedy-mo	13.8	10.6
Greedy-rds	76,206	41,671
Popularity	11.6	10.4
Reads/Size	10.7	10.3
Random	39.5	27.0

Fig. 4. The cumulative distribution function of the client perceived latencies for various algorithms and the respective minimized cost function (in millions) in the table to the right.

system, any point in taking object size into consideration in an RPA. Second, if that is the case, what is currently the best way to represent the object size in an RPA. This is not intended to be a thorough evaluation of the topic, just an illustration of the benefits of this evaluation method.

Figure 4 illustrates the cumulative distribution function of the client-perceived latency for a system with 300 nodes and 10,000 objects in the two leftmost figures, and the respective minimized cost function values in the table to the right. In the Figures, `Greedy-so` refers to problem definition (3), with a P constraint combined with the greedy global heuristic. Thus, this algorithm does not take storage constraints under consideration. If too many objects are allocated to a node, the node will only store the objects that have been accessed most frequently and drop the rest. This is the penalty we assigned for ignoring storage constraints in the RPA. `Greedy-mo` and `Greedy-rds` refer to the same heuristic as the previous algorithm but the problem definitions are (5) and (7) respectively, with a storage constraint. `Popularity` from Table II is a decentralized ranking heuristic that only considers reads and a storage constraint. `Reads/Size` is the same, except that the cost function is $reads_{ik}/size_k$.

There are three main points to take away from Figure 4. First, the minimized cost function values and the results from the CDF graphs differ substantially. Depending on the latency threshold, the relative goodness between algorithms varies a lot. Based on the cost function values, one might think that `Greedy-so` is substantially worse than `Popularity`. But for thresholds above 100 ms for the 25MB storage capacity case, this is not so. The minimized cost function values only show how well a specific heuristic minimizes a cost function, not what performance benefits an RPA would provide in a system. Second, overall, the decentralized `Popularity` and the `Reads/Size` algorithms are the best. Third, algorithm `Greedy-rds` does not work well for our workload and system.

V. Related Work

Despite the importance of the replica placement problem and its practical implications, little has been done to compare the various approaches and their applicability to different system models. A survey of the file assignment problem by Dowdy *et al.* [4] was published in 1982. However, it focuses mostly on problem definitions and their classification and comparison on a feature-by-feature basis. Levin and Morgan [5] introduced a framework for replica placement problems, in the context of distributed databases. They categorize algorithms according to three parameters: whether the workload is static or dynamic; if the information used is partial or complete; whether the problem includes placement of just data or also of programs and computation. Levin and Morgan's work can be seen as a higher-level classification to our framework.

VI. Conclusions

This paper introduces a framework for the classification and evaluation of existing and new replica placement algorithms (RPA). It considers RPAs developed specifically for CDNs, as well as RPAs from other fields that may be applicable to existing or future CDNs. More specifically, the framework can be used in three ways:

- It identifies the qualitative differences of RPAs using a canonical set of primitives that reflect problem definitions and heuristics.
- It uses an analytic model for estimating the decision times of RPAs in large systems. The model takes into account not only computational complexity and message numbers but also disk accesses (due to memory constraints) and message sizes to produce good estimates.
- It allows the comparison of RPAs with different problem definitions, based on the "goodness" of the produced placements.

Using the proposed framework, the paper shows that: (1) Most existing algorithms for CDNs do not scale for systems with more than 10^4 nodes. (2) Algorithms from other fields, such as the file allocation problem, could be applicable to future CDNs, since they take under consideration issues such as writes, update propagation and QoS guarantees (latency and availability); however, they scale even worse that existing CDN algorithms. (3) The evaluation of placement goodness using system performance metrics (e.g., client-perceived latency) results in more realistic and systems-specific comparisons of algorithms, than comparison of minimized cost-function values.

Based on the evaluation of existing approaches, the paper identifies several areas for future research. These include the design of algorithms for systems with constrained resources as well as algorithms that optimize or trade-off multiple goals at the same time. Algorithms that can guarantee QoS properties such as bounded update propagation times and maximum client perceived latency will be very important for future CDNs. Finally, we believe that more effort should be put into decentralized algorithms and decentralized aggregation schemes to address the scalability problem of existing RPAs.

To the best of our knowledge, this is the first attempt to analyze RPAs within CDNs and across a wide spectrum of fields. The aim of this work is to provide a reference point for anyone that wishes to understand, compare and evaluate an instance of the replica placement problem within existing and future CDNs.

References

[1] *Akamai*, Cambridge, MA, USA, http://www.akamai.com.

[2] *Digital Island*, http://www.digitalisland.com.

[3] M. Karlsson and M. Mahalingam, “Do We Need Replica Placement Algorithms in Content Delivery Networks?,” in *Proceedings of the International Workshop on Web Content Caching and Distribution (WCW)*, August 2002.

[4] L. W. Dowdy and D. V. Foster, “Comparative Models of the File Assignment Problem,” *ACM Computer Surveys*, vol. 14, no. 2, pp. 287–313, 1982.

[5] D. Levin and H. Morgan, “Optimizing Distributed Data Bases – A Framework for Research,” in *Proceedings of the AFIPS 1975 NCC*, 1975, pp. 473–478.

[6] B. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann, “Exploiting Locality for Data Management in Systems of Limited Bandwidth,” in *Proceedings of the Symposium on Foundations of Computer Science*, October 1997, pp. 284–293.

[7] S. Mahmoud and J. Riordon, “Optimal Allocation of Resources in Distributed Information Networks,” *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 66–78, March 1976.

[8] R. Tewari and N. Adam, “Distributed File Allocation with Consistency Constraints,” in *Proceedings of the International Conference on Distributed Computing Systems*, 1992, pp. 408–415.

[9] S. Hakimi, “Optimum Location of Switching Centers and the Absolute Centers and Medians of a Graph,” *Operations Research*, vol. 12, pp. 450–459, 1964.

[10] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “On the Placement of Internet Instrumentation,” in *Proceedings of IEEE INFOCOM*, March 2000, pp. 295–304.

[11] S. Jamin, C. Jiu, A. Kurc, D. Raz, and Y. Shavitt, “Constrained Mirror Placement on the Internet,” in *Proceedings of IEEE INFOCOM*, April 2001, pp. 31–40.

[12] B. Li, M. Golin, G. Italiano, and X. Deng, “The Optimal Placement of Web Proxies in the Internet,” in *Proceedings of IEEE INFOCOM*, March 1999, pp. 1282–1290.

[13] M. Korupolu and C. Plaxton, “Analysis of a Local Search Heuristic for Facility Location Problems,” *Journal of Algorithms*, vol. 37, no. 1, pp. 146–188, October 2000.

[14] L. Qiu, V. Padmanabhan, and G. Voelker, “On the Placement of Web Server Replicas,” in *Proceedings of IEEE INFOCOM*, April 2001, pp. 1587–1596.

[15] P. Radoslavov, R. Govindan, and D. Estrin, “Topology-Informed Internet Replica Placement,” *Computer Communications*, vol. 25, no. 4, pp. 384–392, March 2002.

[16] P. Krishnan, D. Raz, and Y. Shavitt, “The Cache Location Problem,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 568–582, October 2000.

[17] M. O’Kelly, “The Location of Interacting Hub Facilities,” *Transportation Science*, vol. 20, pp. 92–106, 1986.

[18] J. Kangasharju, J. Roberts, and K. Ross, “Object Replication Strategies in Content Distribution Networks,” *Computer Communications*, vol. 25, no. 4, pp. 367–383, March 2002.

[19] M. Korupolu, G. Plaxton, and R. Rajaraman, “Placement Algorithms for Hierarchical Cooperative Caching,” *Journal of Algorithms*, vol. 38, no. 1, pp. 260–302, January 2001.

[20] A. Leff, J. Wolf, and P. Yu, “Replication Algorithms in a Remote Caching Architecture,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 11, pp. 1185–1204, November 1993.

[21] A. Venkataramanj, P. Weidmann, and M. Dahlin, “Bandwidth Constrained Placement in a WAN,” in *ACM Symposium on Principles of Distributed Computing (PODC’01)*, August 2001.

[22] I. Baev and R. Rajaraman, “Approximation Algorithms for Data Placement in Arbitrary Networks,” in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms,*, January 2001, pp. 661–670.

[23] M. Balinski, “Integer Programming: Methods, Uses, Computation,” *Management Science*, vol. 12, pp. 253–313, 1965.

[24] I. Cidon, S. Kutten, and R. Soffer, “Optimal Allocation of Electronic Content,” in *Proceedings of IEEE INFOCOM*, April 2001, pp. 1773–1780.

[25] J. Kurose and R. Simha, “A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems,” *IEEE Transactions on Computers*, vol. 38, no. 5, pp. 705–717, May 1989.

[26] O. Wolfson and A. Milo, “The Multicast Policy and Its Relationship of Replicated Data Placement,” *ACM Transactions on Database Systems*, vol. 16, no. 1, pp. 181–205, March 1991.

[27] O. Wolfson and S. Jajodia, “Distributed algorithms for dynamic replication of data,” in *Proc. ACM PODS’92, Symposium on Principles of Database Systems*, June 1992, pp. 149–163.

[28] O. Wolfson, S. Jajodia, and Y. Huang, “An Adaptive Data Replication Algorithm,” *ACM Transactions on Database Systems*, vol. 22, no. 2, pp. 255–314, 1997.

[29] S. Cook, J. Pachl, and I. Pressman, “The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy,” *Distributed Computing*, vol. 15, no. 1, pp. 57–66, 2002.

[30] K. Kalpakis, K. Dasgupta, and O. Wolfson, “Optimal Placement of Replicas in Trees with Read, Write, and Storage Costs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 628–637, June 2001.

[31] C. Krick, H. Räcke, and M. Westermann, “Approximation Algorithms for Data Management in Networks,” in *Proceedings of the Symposium on Parallel Algorithms and Architecture*, July 2001, pp. 237–246.

[32] C. Lund, N. Reingold, J. Westbrook, and D. Yan, “Competitive On-Line Algorithms for Distributed Data Management,” *SIAM Journal of Computing*, vol. 28, no. 3, pp. 1086–1111, May 1999.

[33] M. Fisher and D. Hochbaum, “Database Location in Computer Networks,” *Journal of the ACM*, vol. 27, no. 4, pp. 718–735, October 1980.

[34] K. Chandy and J. Hewes, “File Allocation in Distributed Systems,” in *Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation*, March 1976, pp. 10–13.

[35] F. Meyer auf der Heide, B. Vöcking, and M. Westermann, “Provably Good and Practical Strategies for Non-Uniform Data Management in Networks,” in *Proceedings of the European Symposium on Algorithms*, July 1999, pp. 89–100.

[36] B. Awerbuch, Y. Bartal, and A. Fiat, “Competitive Distributed File Allocation,” in *Proceedings of the ACM Symposium on Theory of Computing*, 1993, pp. 164–173.

[37] B. Awerbuch, Y. Bartal, and A. Fiat, “Distributed Paging for General Networks,” *Journal of Algorithms*, vol. 28, no. 1, pp. 67–104, July 1998.

[38] Y. Bartal, A. Fiat, and Y. Rabani, “Competitive Algorithms for Distributed Data Management (Extended Abstract),” in *Proceedings of the ACM Symposium on Theory of Computing*, 1992, pp. 39–50.

[39] F. Meyer auf der Heide, B. Vöcking, and M. Westermann, “Caching in Networks,” in *Proceedings of the 11th ACM-SIAM Symposium On Discrete Algorithms*, January 2000, pp. 430–439.

[40] M. Karlsson, C. Karamanolis, and M. Mahalingam, “A Framework for Evaluating Replica Placement Algorithms,” Tech. Rep. HPL-2002, HP Laboratories, July 2002, http://www.hpl.hp.com/personal/Magnus_Karlsson.

[41] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, ISBN: 0-262-03141-8. MIT Press, 1989.

[42] J. Current, M. Daskin, and D. Schilling, “Discrete Network Location Models,” in *Facility Location Theory: Applications and Methods*, Z. Drezner and H. Hamacher, Eds., 2001, Forthcoming.

[43] V. Vazirani, *Approximation Algorithms*, ISBN: 3-540-65367-8. Springer-Verlag, 2001.

[44] M. Rabinovich and A. Aggarwal, “RaDaR: A Scalable Architecture for a Global Web Hosting Service,” in *Proceedings of the 8th International World Wide Web Conference*, May 1999, pp. 1545–1561.

[45] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, “A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service,” in *International Conference on Distributed Computing Systems*, May 1999, pp. 101–113.

[46] J. Kubiatowicz et al., “OceanStore: An Architecture for Global-Scale Persistent Storage,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000, pp. 190–201.

[47] C. Krick, F. Meyer auf der Heide, H. Räcke, B. Vöcking, and M. Westermann, “Data Management in Networks: Experimental Evaluation of a Provably Good Strategy,” in *Proceedings of the Symposium on Parallel Algorithms and Architecture*, July 1999, pp. 165–174.

[48] M. Arlitt and T. Jin, “Workload characterization of the 1998 world cup web site,” Tech. Rep. HPL-1999-35R1, HP Laboratories, 1999.

[49] B. Krishnamurthy and J. Wang, “On network-aware clustering of web clients,” in *SIGCOMM*, August 2000, pp. 97–110.

[50] K. Obraczka and F. Silvia, “Network Latency Metrics for Server Proximity,” in *Proceedings of the IEEE Globecom*, November 2000.