A Unified Framework for Evaluating Replica Placement Algorithms

Magnus Karlsson, Christos Karamanolis, and Mallik Mahalingam
Hewlett-Packard Laboratories
1501 Page Mill Road 1134, Palo Alto, CA 94304
Contact Author E-mail: karlsson@hpl.hp.com

Abstract

The placement of data to maximize the performance and minimize the cost of a computing system is an optimization problem that has been studied extensively in several fields, including distributed databases, storage systems and, more recently, content delivery networks. However, little has been done to compare the various approaches and their applicability to different systems.

This paper introduces a framework that facilitates the evaluation of replica placement algorithms. Existing algorithms are mapped to a generic set of primitives describing both problem definitions and heuristics. Based on these primitives, contributions of proposed algorithms can be crisply identified and reported. The primitives are also used to estimate the time it takes for an algorithm to reach a decision. Using actual implementations, we show that message sizes and disk access also have to be taken into account to provide a good estimate, not only the traditional computation and message complexities. We show that most proposed algorithms are prohibitively expensive for systems with more than 10^4 *nodes and* 10^4 *objects. We also report on comparisons between algorithms with different system models, based on the goodness of the produced placement.*

1 Introduction

This paper proposes a unified framework that facilitates the evaluation of replica placement algorithms. *Replica placement algorithms* decide what data to replicate on what storage nodes, and are a key technology in Content Delivery Networks (CDN), such as Akamai [1] and Digital Island [12]. A number of replica placement algorithms have been proposed in the CDN literature, but until now, there has been no systematic way to classify and compare these algorithms.

The replica placement problem has also been studied extensively in several other distributed computing fields, including the file assignment problem (FAP) in storage systems [13], distributed databases [26], network location [11], and data management [29]. So, a question that arises is how all these approaches are related to each other and how can one systematically represent and evaluate new as well as existing algorithms within and between these fields.

In this paper, we propose a framework that facilitates the qualitative and quantitative evaluation of replica placement algorithms, existing and new ones. When proposing a new algorithm two things have to be performed and reported: pin-point the fundamental differences with prior art, and evaluate the new algorithm against prior art. The two important evaluation metrics are (1) how long it takes for the algorithm to reach a decision (*decision time*), and (2) how "good" the produced placement is.

We define a placement *algorithm* to consist of two parts: a *problem definition* and a *heuristic*. The problem definition consists of a cost function to be minimized under some constraints. Most of these formulations result in NP-complete problems [15]. Thus, heuristics are used to produce near-optimal solutions. Section 2 presents a generic set of primitives that can represent algorithms from various fields. Algorithms, from a wide range of systems, are decomposed into these primitives to achieve three purposes. First, report the fundamental differences between existing algorithms, within and between different distributed systems fields. Second, identify and report the differences between a new algorithm and prior art. Third, if the new problem definition has already been proposed, our paper identifies heuristics that have been used for that problem; it also identifies other problem definitions have been used together with a heuristic.

The first important metric for use in comparing algorithms is concerned with the time it takes to reach a placement decision. Traditionally, only computation complexity and number of messages sent are considered. In Section 3, we use actual algorithm implementations to show that message sizes and disk access time due to memory space constraints also need to be taken into account to provide a good estimate. The decision times of proposed algorithms can be estimated by decomposing them into our set of primitives and corresponding equations. We show that most proposed algorithms are prohibitively expensive for systems with more than 10^4 clients, nodes and objects which makes them unsuitable for use in a global file service.

The other important evaluation metric is related to the

goodness of a produced placement. A problem with the literature is that algorithms with different problem definitions that target the same goal and system, have not been compared as this is not possible using traditional minimized cost function values. One way of solving this is to compare their respective impact on the *overall goal* of the replica placement problem. This is defined as either meeting the performance goals of clients with the minimum possible infrastructure cost, or maximizing the client perceived performance given an infrastructure. In Section 4, we illustrate this by comparing a number of previously proposed and new algorithms with various problem definitions for use in CDNs. We show that for a real-life workload, it makes a big difference to take access frequencies into account. Simpler algorithms, that do not consider access frequencies, are often not any better than a pure random placement. Finally, we present related work in Section 5 and conclude in Section 6.

2 Replica Placement Algorithms

To be able to achieve a performance or cost improvement, a *problem definition* is formulated that approximates the overall goal, the workload and the system in question. These problems are usually NP-complete [15] thus requiring a *heuristic* to be able to find an approximate solution within feasible time. In this paper, *algorithm* refers to a heuristic applied to a specific problem definition. Our unified framework abstracts the commonalities among existing replica placement algorithms and parameterizes the problem definitions in Section 2.2 and heuristics in Section 2.3 into primitives. This can be used to find the fundamental differences between new problem definitions and heuristics and previous ones. If the problem definition already exists, the references in Table 1 (or Table 2 in the next Section) can be consulted to find suitable heuristics for the problem.

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. In particular, we are interested in algorithms that are applicable to Internet-like topologies. Thus, we only include algorithms that are applicable to general or tree topologies.

2.1 System Model and Overall Goal

The system considered in this paper is a storage repository consisting of a set of storage nodes, hereafter just called *nodes*, interconnected with some links. On these nodes, objects are stored representing data aggregates such as files, web-sites or volumes. The storage nodes have clients connected to them that generate accesses to the objects.

The *overall goal* of the replica placement problem is to decide the location of object replicas in the above system as to *maximize either the client perceived performance*[1] *given an existing infrastructure, or minimize the infrastructure cost given a client perceived performance goal*. This overall goal is always abstracted into a problem definition that is used as an optimization goal.

[1] Performance could here be e.g., latency, throughput or availability.

2.2 Problem Definition and its Primitives

Replica placement problem definitions consists of one fixed part and one variable part that we have broken down into primitives. The problem can be formally stated as follows. Given a set of clients C, nodes N, objects K, and links L, the sizes of these sets being $|C|$, $|N|$, $|K|$ and $|L|$, respectively. If there is a physical communication channel between two nodes, a link must be added between these nodes. Each client $i \in C$ must be assigned to a node $j \in N$ for each object $k \in K$, incurring a specific cost according to a *cost function*, described later. To this, *constraints* are added. Four constraints that are usually present are: (1) each client can only send requests for an object to one node, (2) only nodes that has the object can respond to requests for it, and (3) plus (4) that objects and requests cannot be split. Other additional constraints that might be added to this will be described later. Here, y_{ijk} indicates whether client i sends it requests for object k to node j, and x_{jk} indicates whether node j stores object k. The problem is then to find a solution of either minimum cost, when it refers to response times and dollar cost, or maximum cost when it refers to throughput that satisfies all the constraints as stated below.

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

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

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

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

We have found that there are six primitive parameters that have been considered in cost functions of the replica placement problem. We believe that this set, and the set of additional constraints, are probably complete, as after examining 40% of existing problem definitions no more new primitives had to be added.

Reads ($reads_{ik}$) : The number of read accesses by a client i to an object k per time unit. 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 number of write accesses by a client i to an object k per time unit.

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

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

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

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

The constraint primitives that can be added to the basic constraints of the problem definition are:

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

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

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

Link Capacity (CL_l) $\sum_{i \in C} \sum_{k \in K} (reads_{ik} + writes_{ik}) \cdot size_k \cdot y'_{lik} \leq CL_l, \forall l)$: A bandwidth constraint on the link between two nodes. y'_{ilk} is one if accesses for object k from client i uses link l, otherwise it is zero. Some problem definitions separate reads, writes and 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 a given j and k : The location of an object source copy is specified.

Delay (D) : Specifies the desired minimum delay of requests in the system. The mathematical definition is found in [30] and is omitted here due to space constraints.

Availability (AV) : Specifies the desired minimum availability of objects in the system. The mathematical definitions are found in [30, 39] and are omitted here due to space constraints.

Table 1 shows previously proposed replica placement problem definitions from many disparate fields mapped into our framework. The problem definitions have been broken down into three major groups according to what object accesses (if any) that they take into account. The first group ignores any accesses, while the second group only accounts for read accesses, and the third group considers both read and write accesses including the consistency requirements of the latter. These groups are further divided into four categories according to whether a problem definition takes *single* or *multiple objects* into account, and whether it considers *storage costs*. The drawback with single-object problem formulations is that they cannot be used with constraints that are intra-object dependent, such as a storage constraint (SC). The advantage is that they are easier to solve.

Group 1 problem definitions are useful when placing one or a few objects as they are relatively easy to solve, but they will place all objects in the same nodes if the number of replicas constraint is fixed. The two problem definitions of Group 1 are k-center problems [16], a subset of network location problems [11].

Group 2 problem definitions are used for read-only and read-mostly workloads, mainly in web related areas such as content delivery networks (3), (5), (6), (8), (9) and cache location problems (4). This group consists of many classical network location problems such as (3), (5) the *k-median problem*, (4) the *hub location problem*, and (8), (9) the *facility location problem* or the *fixed charge location problem*. The main difference between the cost functions within a group is how they interpret distance. In a CDN, distance is generally measured between the client and the closest node that has a copy of the object; while for a cache, it is the distance between the client and the cache plus, the distance between the cache and the node for all cache misses.

Group 3 contains problem definitions from distributed databases (10), (12), (13), the file allocation problem in storage systems (14), (15), (18), (19) and data management (11), (12), (16), (17). There are two main flavors of cost functions in Group 3: the ones that minimize the maximum link congestion (11), (16), (17), and the ones that minimize the average client access latency or other client costs. Another difference between the problem definitions is how they represent the cost for keeping objects consistent when written. 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) and (16) - (18), the distribution distance is the minimum spanning tree. In (13) this is the sum of the distances between the accessed node and the other nodes with a copy of the object. Problem definition (14) does not specify how this cost is calculated, and (15) does not specify how to calculate the distance for writes. Problem definition (19) is quite different from the others and we refer to the source paper [30] for further details.

2.3 Heuristic Primitives

In order to solve the NP-complete problem defined in the previous section, a heuristic is used. By decomposing a heuristic into our set of primitives, different heuristics can be systematically compared and contrasted. We have found that heuristics can be characterized by three primitives: metric scope, approximation method and cost func-

#	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	[16, 17, 18]
(2)	$\sum_{i \in C} \sum_{j \in N} dist_{ij} \cdot y_{ijk}$	P	[16, 18]
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	[16, 18, 27, 35, 38]
(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	[24, 34]
	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	[20, 21]
(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	[25, 41]
(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	[4]
	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	[5, 8]
	Multiple Objects + Storage Cost		
(9)	$\sum_{i \in C} \sum_{j \in N} \sum_{k \in K} (sc_{jk} \cdot x_{jk} + dist_{ij} \cdot reads_{ik} \cdot y_{ijk})$	-	[8]
Group 3: Considers both read and write accesses including consistency requirements of the latter.			
	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, SC	[42]
(11)	$\max_{l \in L} \sum_{i \in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) / CL_l$	-	[29]
	Single Object + Storage Cost		
(12)	$\sum_{i \in C} \sum_{j \in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot (dist_{ij} + mst_{jk}) \cdot y_{ijk} + \sum_{j \in N} sc_{jk} \cdot x_{jk}$	P, SC	[19, 23, 28]
(13)	$\sum_{i \in C} \sum_{j \in N} (reads_{ik} \cdot dist_{ij} + writes_{ik} \cdot (dist_{ij} + \sum_{n \in N, l \neq j} dist_{jn} \cdot x_{nk})) \cdot y_{ijk} + \sum_{j \in N} sc_{jk} \cdot x_{jk}$	P, SC	[14]
(14)	$\sum_{j \in N} (\sum_{i \in C, i \neq j} reads_{ik} \cdot dist_{ij} \cdot y_{ijk}) + (sc_{jk} + update_cost_{jk}) \cdot x_{jk}$	-	[7]
(15)	$\sum_{i \in C} \sum_{j \in N} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot update_dist_{ij} \cdot availability_{ijk} \cdot x_{jk} + \sum_{j \in N} sc_{jk} \cdot x_{jk}$	AV	[39]
(16)	$\max_{l \in L} (\sum_{i \in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) + \sum_{j \in N} sc_{jk} \cdot y^m_{jlk}) / CL_l$	-	[28]
(17)	$\max_{l \in L} (\sum_{i \in C} (reads_{ik} \cdot y^r_{ilk} + writes_{ik} \cdot y^w_{ilk}) + \sum_{j \in N} sc_{jk} \cdot y^m_{jlk} + info_exchange_{lk}) / CL_l$	-	[31, 32]
	Multiple Objects + Storage Cost		
(18)	$\sum_{j \in N} \sum_{k \in K} \sum_{i \in C} reads_{ik} \cdot dist_{ij} \cdot y_{ijk} + writes_{ik} \cdot (dist_{ij} + mst_{jk}) \cdot y_{ijk} + \sum_{j \in N} \sum_{k \in K} sc_{jk} \cdot x_{jk}$	-	[3, 6]
(19)	$\sum_{i \in C} \sum_{l \in L} \sum_{k \in K} channel_cost_l \cdot y'_{ilk} + \sum_{j \in N} \sum_{k \in K} sc_{jk} \cdot size_k \cdot x_{jk}$	CL, D, AV	[30]

Table 1: 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 included here.

tion simplification.

Metric Scope. The set of clients, nodes, objects and links that is considered when making a placement decision. The heuristic can set this scope anywhere from none to all. The more clients that are taken into account, the more messages have to be sent. This generally means that more storage space is consumed, and thus it will take longer for the algorithm to complete. If only one node is considered, the heuristic is decentralized and has to be run everywhere, but if all nodes are considered, it is centralized. If only local object knowledge is specified, only the objects stored in the node or objects that have been referenced by the node are considered.

Approximation Method. Within the specified metric scope, the heuristic has to pick a subset of feasible solutions and find the best one within that set. This is accomplished by the use of an approximation method. The methods used in this area so far are:

Ranking ($R(greedy)$) Compute the cost impact of all possible combinations of placing one extra object on one node. Sort these costs in ascending or descending order dependent on the cost function, pick the top one and place it if none of the constraints present are violated. If they are violated, try the next one in the list. A *greedy* ranking algorithm [9] can be constructed by recomputing the cost function after each object that is placed. Ranking is generic and can be used for all problem definitions.

Fixed Threshold ($T(threshold)$) An object is placed at a specific node if the cost function savings due to this placement exceeds a specified threshold. This approximation method is usually independent of the problem definition.

Improvement ($I(method)$) Contains methods that tries to improve on a solution by introducing small changes.

The m-*distance* improvement heuristic [10] evaluates all possible permutations of the solution, or a subset thereof, where at the most m bits in the decision variables are interchanged. There are numerous other interchange methods, many of them referenced in [10]. The improvement methods are independent of the problem definition.

Relaxation Techniques ($Re(method)$) Encompasses a number of techniques. *Lagrangian relaxation* [40] relaxes the constraints of the problem definition by moving them to the cost function. This technique provides a bound on how far from optimal a solution is. *Linear relaxation* [40] relaxes the integer constraints of the original problem. It is now 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) Dynamic programming [9] uses the fact that the same calculations are often repeated many times. These intermediate results are saved and later reused. In some cases, rules are also applied in order to merge intermediate results. Have predominately been used for tree topologies and are specific to the problem definition.

Parametric Pruning (PP) Parametric pruning [40] is a technique of pruning irrelevant portions of the search space. To perform this pruning, the optimal cost needs to be known. As this is generally not known, optimal cost guesses are computed and the pruning made using these. Problem definition specific.

Hierarchical (H) All of the above approximation methods can be used in a hierarchical fashion. For example, ranking can be used on each node on the leaf level. The results from the ranking are then aggregated by the nodes on the next higher level that performs another ranking using this aggregated data, and so on up to the root node [41].

Multi-phase ($M(method_1, method_2, ...)$) Above mentioned techniques can be used in conjunction with each other. E.g., greedy ranking followed by an improvement heuristic [18, 41]. Multi-phase can also be combined with hierarchies [41].

Cost Function Simplifications. Simplifications (or modifications) made to the original cost function of the problem definition. E.g., 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 the Hotspot heuristics disregards the distance and uses just $\sum_{i \in C} \sum_{j \in N} reads_{ik} \cdot y_{ijk}$.

Table 2 lists previously proposed heuristics for replica placement problems, mapped into our framework. The table serves mainly two purposes. The first is to qualitatively compare existing heuristics that have been used. The main observations are that problem definitions from group 1 and 2 have used many generic heuristics, while group 3 has not (with the exception of problem definition 13). Another observation is that many of the ranking heuristics are quite similar. The second purpose is that the contribution of a newly proposed heuristic can be crisply identified and reported. Note that the table and the primitives only give a high-level view of the heuristics, and in order to faithfully implement one the source reference has to be consulted. For example, while the tree-based heuristics are grouped together in Table 2, their respective implementations for the six problem definitions are different. Heuristics from [28, 29, 31, 32] are not included due to space constraints. The link metric scope is not shown in the table as none of the heuristics use it.

Table 2 indirectly reflects if a heuristic is an *off-line* or an *on-line* approach. An off-line heuristic would start from scratch every time it is executed and make the placement decision for all objects and nodes it considers. The deletion of objects is implicit as the heuristic only decides on what nodes to place an object. Thus, all the other nodes that the heuristic stated did not have the object, simply deletes that object if they have it. An on-line approach, on the other hand, makes small partial changes to the placement as time goes by. For an on-line heuristic, the deletion of objects have to be done explicitly. This means that there is one heuristic for placing objects and another one for deleting them as can be seen in the table for RaDaR, Awerbuch FAP, and Dist Awerbuch.

3 Decision Time

Decision time is defined as 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 the decision time. In this section, we advocate that, in addition to the aforementioned components, message sizes and disk accesses also have to be taken into account to provide a good estimate. We also develop a model that, using the primitives of the previous Section, estimates the decision time of many algorithms

The *computational cost* is the time required to reach a decision, in a node, given that all necessary data are already in the primary memory of the node. It is the computational complexity (C_{comp}) of the algorithm times the average time it takes to perform one iteration or calculation (T_{comp}). The *message cost* is the time spent on sending and receiving messages; it does not include message transfer time – we assume that useful computations can be scheduled while messages are in transit. It consists of two components: one that is proportional to the number of messages (C_{msg}), where T_{msg} is the time for sending and re-

Heuristic	Approximation Method	Metric Scope			Cost Function Simplification	Used for
		Client	Node	Object		
Greedy Global [20, 35]	R(yes)	all	all	all	-	3,5
Ranking Local [20, 25]	R(no)	local	one	local	-	5,6
Popularity [20]	R(no)	local	one	local	$reads_{ik}$	5
Hotspot [35]	R(no)	vicinity	all	all	$reads_{ik}$	3
Fan-Out Based[18, 38]	R(no)	indep	all	indep	$fanout_j$	1,2,3
2-distance [7]	I(2-dist)	all	all	all	-	14
l-Greedy [18]	M(R(yes),I(m-dist))	all	all	indep	-	1,2
Ranking Dist [25]	R(no)	local	one	local	$dist_{ij}$	6
Hierarchical [21, 41]	H M(R(no),I(2-dist))	child	child	child	$dist_{ij} = \max_{i,j} dist_{ij}$	5,6
RaDaR [36, 37]	Place: T($reads_{ik} / \sum_{i \in C} y_{ijk} > C$)	local	two	local	?	?
	Delete: T($\sum_{i \in C} y_{ijk} < 1$)	local	one	local	?	?
Awerbuch FAP [3, 6]	Reads: T($reads_{ik} > C$)	all	all	all	$reads_{ik}$	18
	Writes: R(no)	all	all	all	$\sum_{i \in Writers} dist_{ij}$	18
Dist Awerbuch [3, 6]	Reads: T($reads_{ik} > C$)	local	vicinity	local	$reads_{ik}$	18
	Writes: R(no)	local	writers	local	$\sum_{i \in Writers} dist_{ij}$	18
Min k-Center [17, 18]	PP	all	all	indep	$dist_{ij}$	1
Tree-based [8, 19, 24, 27, 35, 42]	DP	all	all	all	-	3,4,8,9, 10,12
Lin [4]	RE(linear)	all	all	all	-	3,5,7,8,9
Lin+Imp [39]	M(RE(linear),I(drop),I(add/drop))	all	all	all	-	15
Lagr [14]	RE(lagrangian)	all	all	all	-	3,5,13
Lagr+Imp [30]	M(RE(lagrangian),I(add/drop))	all	all	all	-	19

Table 2: Existing replica placement heuristics mapped into our unified framework. A "?" means that it is not clear from the paper what this entry is. "Child" means that the metric scope is that of the children to a node in the tree. "Vicinity" means a number of entities close to a node (see the corresponding paper for details). The "used for" column lists problem definitions that this heuristic has been applied to in the literature.

ceiving 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 (S_{tot}) minus the local data-set on the node (S_{local}) times T_{msgpb}, the time spent per byte of transmitted data. The *storage cost* is the time required for accessing data on the disk, when they do not fit in the node's primary memory. is the amount memory required to make the decision (S_{req}) minus the effective amount of memory in the node (M). This is multiplied by the number of times this data is reused before the decision is made (C_{disk}) and the time it takes to read one byte of data from disk (T_{diskpb}). The total decision time is then the sum of the individual components:

$$T_{tot} = C_{comp} \cdot T_{comp} + C_{msg} \cdot T_{msg} + (S_{tot} - S_{local}) \cdot T_{msgpb} + \max(S_{req} - M, 0) \cdot C_{disk} \cdot T_{diskpb}$$

3.1 Decision Time of Existing Heuristics

We demonstrate that computation, message and disk costs all have to be taken into account in order to provide a good estimate, by using our model on existing algorithms. We use problem definition (3) from Table 1 with a number of replica constraints.

For the purposes of this evaluation, we have developed a general placement-algorithm tool, called `Coeus`. Given the properties of an algorithm, according to the framework of section 2, `Coeus` produces the placement of that algorithm. When possible, we use good algorithm implementations published in the literature. When none is available, we try to provide an implementation which is as optimized as possible.

The measured algorithm parameters, shown in the top of Table 3, were gathered in the following way. First, we measured T_{comp} for each of the algorithms and T_{diskpb} for one approximation method (used by all the others) using `Coeus`. The measurements were performed on a 552MHz PA8600 processor. Full compiler optimizations were used and the entire data set was kept in memory for the T_{comp} measurements; it was larger than the available memory when measuring T_{diskpb}. T_{msg} and T_{msgpb} were chosen so that a node can receive 10^3 10K-messages per second, a value typical for web servers.

The computational complexities (C_{comp}) in Table 3 are deduced by studying the C-code of the implementations or by just getting the computational complexities from the source papers. S_{tot} is calculated by adding up the number of variables the node requires to make a decision. This number is multiplied by eight (assuming 64-bit variables) to produce the number of bytes. S_{local} is calculated in the same way, but only for data gathered at the local node.

To validate the decision time model, the parameter values from Table 3 are used. The size of the system was varied between 100 clients and nodes to $2 \cdot 10^4$ and the individual decision times of the analytical model is compared with the individual times of the real implementation. This comparison shows that the estimates from the model are between 0.5 and 2 times the decision times of the real implementations. This might sound excessive, but as we will see later

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(yes)	$CNRK$
C_{comp} R(no)	CNK	C_{comp} I(2-dist)	$I_{2dist}CNK$	C_{comp} Re(Lagr)	$I_{lagr}CNRK$
S_{req} R(yes)	$8(NCu_{dist}+Cu_{dem})$	S_{req} R(no)	$8(NCu_{dist}+Cu_{dem})$	S_{req} I(2-dist)	$8(NCu_{dist}+Cu_{dem})$
S_{req} Re(Lagr)	$8(NC+2C+4N)$	C_{disk} R(yes)	RK	C_{disk} R(no)	K
C_{disk} I(2-dist)	$K\sqrt{I_{2dist}}$	C_{disk} Re(Lagr)	KI_{lagr}		

Table 3: 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.

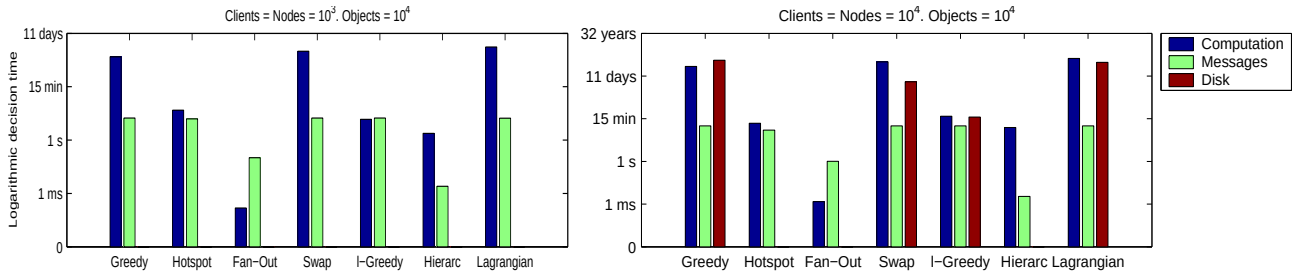

Figure 1: The decision costs for some existing heuristics at medium scales. $I_{2dist} = 10^4$ and $I_{lagr} = 115$.

the differences between the computation, message and disk components, and between algorithms, are several orders of magnitude, dwarfing the error of our model.

Figure 1 depicts the decision costs for seven heuristics from Table 2. These were chosen because they represent a broad spectrum of heuristics. The numbers of nodes and clients are 10^3 and 10^4, 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 message cost, and others by disk cost, signifying the importance to take all three factors into account. For example, someone only looking at the computation time for the greedy heuristic for 10^4 nodes might conclude that it is feasible, while it will take several weeks longer due to disk accesses. With our model, disk accesses are correctly estimated to be the dominant decision time factor. The dominant factor for a heuristic may change as the system scale and problem definition varies (e.g., *greedy* between the two graphs). Second, the difference between various heuristics' decision times is very large. Third, even at this small scale, some heuristics are infeasible as they take weeks to terminate.

Figure 2 shows the decision costs of a system with 10^6 client and nodes and 10^8 objects: the size of a typical global file system, for example. Due to its size, we cannot measure the I variables as in Figure 1. Instead, we assume that $I_{2dist} = 10^8$ is enough when seeded with a placement from some other heuristic. We believe that using I(2-dist) on its own for such a scenario without seeding it with a good placement is not meaningful. In the case of the Lagrangian relaxation algorithm, we assume that it terminates after 300 iterations.

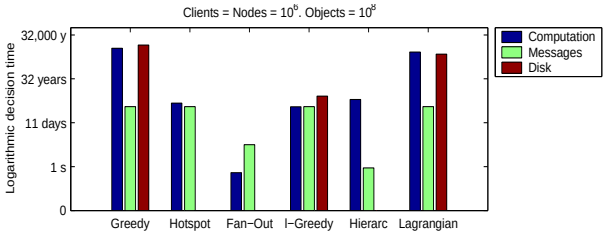

Figure 2: The decision costs for six heuristics at global file system scales.

For these scales, not a single 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.

3.2 Decision Time of New Algorithms

If a new heuristic is proposed for the previous problem definition, the decision time of that algorithm can be estimated by using Table 3. To achieve that, find the approximation method used in the table, input the metric scopes of the new heuristic, and calculate the decision time. If the problem definition itself is changed or there is a cost function simplification the effects on decision time is more intricate. The problem definition can be varied along four axes:

the cost function can take into account single or multiple objects; constraints can be added or removed; the cost function can be taken either from group 1, 2 or 3 (see Table 1); finally, the cost function can consider storage cost or not. How changes along these axes affect the individual components of the decision time depends on the approximation method used. For example, adding a constraint can decrease C_{comp} for R(yes), increase it for Re(Lagr), and has no effect on R(no). The details are reported in the Appendix.

4 Goodness of Placement

In the literature, the goodness of a certain heuristic in solving a specific problem definition is evaluated using competitive ratios or comparisons of minimized cost function values. However, in many research fields such as CDN, FAP, and distributed databases, algorithms have been proposed that use different problem definitions for the same system and overall goal. The problem is that the goodness of the placements of these algorithms have not been compared against each other, even though they target the same system and goal; this is not possible using the aforementioned methods. We propose a solution to this, by quantitatively evaluating an algorithm's impact on the overall goal of the replica placement problem (see Section 2.1). This is useful even in the context of a single problem definition as shown in [22], as it reflects how well the problem definition corresponds to the system and overall goal.

An overall goal related to the dollar cost of the system is straight forward to measure. Evaluating performance goals is harder. First, the definition of performance depends on what the system is used for and what is important for the user. For example, performance could be the minimum total response time, maximum system throughput, or maximum data availability. Second, when comparing the placement produced by algorithms with different problem definitions, it is important to factor the penalty of not meeting certain constraints or taking certain parameters into the evaluation. When the performance impact of a placement is measured on an actual system, such factors are reflected on the result. However, when the performance in evaluated using analytical models or simulators, the constraints 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 algorithms for use in CDNs [2]. These algorithms have never been compared before, as they use different problem definitions. We base our evaluation on a system simulation and real web-access traces.

4.1 Experimental Methodology

Three ingredients are required to evaluate performance-based placement goodness: a performance metric, a representative workload, and the placements generated by the various algorithms. The *placement* for the evaluations in this section are produced by `Coeus`, described in Section 3.1. The other two issues are discussed in the following paragraphs.

In practice, the satisfaction of the clients accessing a web service is expressed by some metric's *threshold value*. For example, service level agreements (SLA) used in CDNs are of the form *the Xth percentile of requests have a response latency below Y msec*. Performance above some threshold is unacceptable by the users. For our study, we use a latency threshold as the 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 [2] to generate a realistic workload for a CDN. To reduce the client population to a tractable size, we used the log of the busiest day and clustered all encountered client IP addresses according to the Autonomous System (AS) they belong to. This clustering reduce the number of clients from 2.6M to 5.4K. These clusters represent both the clients in the system and the nodes on which object may be placed. Matrix $reads_{ik}$ is obtained by counting the number of requests each such node generates. To simulate systems with less than 5.4K nodes (e.g., 1000), we rank the nodes according to $reads_{ik}$; we then choose the desired number of nodes in a way that preserves the original distribution of $reads_{ik}$. Each web page is treated as a separate object. There are 89K objects, which are reduced to the desired number by aggregating them according to the formula: $obj_id_{new} = obj_id_{old} \cdot K_{desired}/K_{total}$

In an ideal world, matrix $dist_{ij}$ would represent the average latency between nodes i to j. However, this is impossible to measure, unless one happens to be on that specific routing path. 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 [33]. To turn these hop numbers into latencies with some variation (as the actual latencies undoubtedly have), we use the formula $Latency = |200 \cdot (hops + \Delta)|\ ms$, where Δ is a random value in $[-0.5..+0.5]$. $dist_{ij}$ is made up of the nodes picked when creating $reads_{ik}$.

4.2 Goodness of Existing Algorithms

Figure 3 illustrates the cumulative distribution function of the client-perceived latency for a system with 1000 nodes, 100 objects and 32 replicas of each object. We focus on three problem definitions used in CDNs (total distance · demand, total distance and max distance) and a number of previously proposed heuristics taken from Table 2. There are three main points to take away from the graphs. First, for the greedy, swap and lagrangian heuristics, taking demand (reads) under consideration makes a big difference; especially, for a low threshold (e.g., 200ms). Second, Swap and Hotspot are not much better than Random for the max

[2] This is not intended to be an exhaustive list of algorithms.

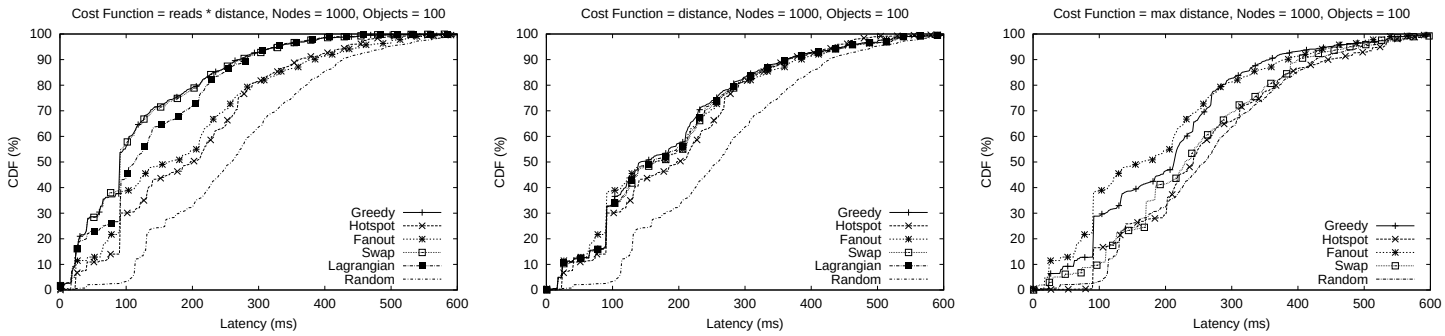

Figure 3: The cumulative distribution function of the client perceived latencies for various algorithms.

distance problem definition. Third, comparing minimized cost function values would give a misleading picture. For example, for algorithms using a total distance cost function, the cost function value differ with around 10%. Our results (graph 2) show a much smaller difference. Even more impressively, for the max distance cost function the results are completely inverted. In terms of cost functions, swap has the lowest cost, but according to our results, it is among the worst (worst or second worst depending on the threshold).

5 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.*[13] was published in 1982. However, it focuses mostly on problem definitions and their classifications and comparison on a feature by feature basis. Levin and Morgan [26] 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.

6 Conclusions

This paper introduces a framework that facilitates the evaluation of replica placement algorithms. First, problem definitions from a wide range of fields, including file allocation problem, distributed databases and content delivery networks are reduced to a canonical set of cost function parameters and constraints. Similarly, heuristics are classified according to three parameters: metric scope, approximation method and cost-function simplifications. This classification illustrates (1) how problem definitions reflect important aspects of the system model, (2) that certain formulations are applicable to multiple problem spaces, and (3) what heuristics are applicable to different problem definitions and system scales.

Second, algorithms should be compared against each other on the basis of how long it takes to produce the placement and how good the produced placement is. Regarding the first comparison, we claim that pure computational cost and message numbers are not sufficient to capture the decision time of an algorithm. The cost for disk access (due to memory constraints) and for message sizes have also to be taken into account. We use actual algorithm implementations to demonstrate the contribution of the four components to the decision time. We show that most existing algorithms do not scale for systems with 10^4 clients, nodes and objects. Based on this analysis, we provide rules of thumb for the proposal of new algorithms. That is, how the problem definition and heuristic affect the various components of the decision time.

Regarding the goodness of the placement, we propose comparing algorithms by evaluating their impact on the overall goal of the system. This can be either a performance metric or a dollar cost. We use actual implementations of algorithms for CDNs to demonstrate that the achieved goodness of each algorithm differs significantly from the minimized cost functions values that are used traditionally. Moreover, we demonstrate that this approach makes cross-problem definition comparisons possible. This is necessary when multiple problem definitions have been proposed for the same overall goal and system.

To the best of our knowledge, this is the first attempt to analyze replica placement algorithms, across a wide spectrum of fields. The aim for this work is to provide a reference point for anyone that addresses an instance of the replica placement problem. That is, assist in understanding, comparing and evaluating placement algorithms.

References

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

[2] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-1999-35R1, HP Laboratories, 1998.

[3] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive Distributed File Allocation. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 164–173, 1993.

[4] 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,*, pages 661–670, January 2001.

[5] M. Balinski. Integer Programming: Methods, Uses, Computation. *Management Science*, 12:253–313, 1965.

[6] 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*, pages 39–50, 1992.

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

[8] I. Cidon, S. Kutten, and R. Soffer. Optimal Allocation of Electronic Content. In *Proceedings of IEEE INFOCOM*, pages 1773–1780, April 2001.

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

[10] J. Current, M. Daskin, and D. Schilling. Discrete Network Location Models. In Z. Drezner and H. Hamacher, editors, *Facility Location Theory: Applications and Methods*, 2001. Forthcoming.

[11] M. Daskin. *Network and Discrete Location: Models, Algorithms and Applications*. ISBN: 0-471-01897-X. John Wiley & Sons, Inc., April 1995.

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

[13] L. W. Dowdy and D. V. Foster. Comparative Models of the File Assignment Problem. *ACM Computer Surveys*, 14(2):287–313, 1982.

[14] M. Fisher and D. Hochbaum. Database Location in Computer Networks. *Journal of the ACM*, 27(4):718–735, October 1980.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[16] S. Hakimi. Optimum Location of Switching Centers and the Absolute Centers and Medians of a Graph. *Operations Research*, 12:450–459, 1964.

[17] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the Placement of Internet Instrumentation. In *Proceedings of IEEE INFOCOM*, pages 295–304, March 2000.

[18] S. Jamin, C. Jiu, A. Kurc, D. Raz, and Y. Shavitt. Constrained Mirror Placement on the Internet. In *Proceedings of IEEE INFOCOM*, pages 31–40, April 2001.

[19] 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*, 12(6):628–637, June 2001.

[20] J. Kangasharju, J. Roberts, and K. Ross. Object Replication Strategies in Content Distribution Networks. In *Proceedings of the International Workshop on Web Caching and Content Distribution*, June 2001.

[21] G. Korupolu, M. ad Plaxton and R. Rajaraman. Placement Algorithms for Hierarchical Cooperative Caching. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 586–595, January 1999.

[22] 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*, pages 165–174, July 1999.

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

[24] P. Krishnan, D. Raz, and Y. Shavitt. The Cache Location Problem. *IEEE/ACM Transactions on Networking*, 8(5):568–582, October 2000.

[25] A. Leff, J. Wolf, and P. Yu. Replication Algorithms in a Remote Caching Architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1185–1204, November 1993.

[26] D. Levin and H. Morgan. Optimizing Distributed Data Bases – A Framework for Research. In *Proceedings of the AFIPS 1975 NCC*, pages 473–478, 1975.

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

[28] C. Lund, N. Reingold, J. Westbrook, and D. Yan. Competitive On-Line Algorithms for Distributed Data Management. *SIAM Journal of Computing*, 28(3):1086–1111, May 1999.

[29] 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*, pages 284–293, October 1997.

[30] S. Mahmoud and J. Riordon. Optimal Allocation of Resources in Distributed Information Networks. *ACM Transactions on Database Systems*, 1(1):66–78, March 1976.

[31] 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*, pages 89–100, July 1999.

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

[33] K. Obraczka and F. Silvia. Network Latency Metrics for Server Proximity. In *Proceedings of the IEEE Globecom*, November 2000.

[34] M. O'Kelly. The Location of Interacting Hub Facilities. *Transportation Science*, 20:92–106, 1986.

[35] L. Qiu, V. Padmanabhan, and G. Voelker. On the Placement of Web Server Replicas. In *Proceedings of IEEE INFOCOM*, pages 1587–1596, April 2001.

[36] 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*, pages 1545–1561, May 1999.

[37] 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*, pages 101–113, May 1999.

[38] P. Radoslavov, R. Govindan, and D. Estrin. Topology-Informed Internet Replica Placement. In *Proceedings of the International Workshop on Web Caching and Content Distribution*, June 2001.

[39] R. Tewari and N. Adam. Distributed File Allocation with Consistency Constraints. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 408–415, 1992.

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

[41] 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.

[42] O. Wolfson and A. Milo. The Multicast Policy and Its Relationship of Replicated Data Placement. *ACM Transactions on Database Systems*, 16(1):181–205, March 1991.

Appendix

When the actual problem definition changes or a cost function simplification is introduced, consult Table 4 that summarizes the relationship between problem definition primitives and the various components that make up the decision time. A $\uparrow$ indicates that the value of the metric

Problem Def. Change	For All	Param	R(yes)	R(no)	I(2-dist)	Re(Lagr)	T()
single obj $\rightarrow$ multiple	T_{comp} -	C_{comp}	$\uparrow$ as R grows	$\uparrow$ due to sorting	$\uparrow$ as I_{2dist} increases	$\uparrow$ as I_{lagr} and R increases	$\uparrow$
	S_{tot} -	C_{disk}	-	$\downarrow$ to I_{sort}	$\uparrow$ as I_{2dist} increases	$\uparrow$ as I_{lagr} and R increases	$\uparrow$
		S_{req}	$\uparrow 8(NCu_{dist} + CKu_{dem})$	$\uparrow 8(NCu_{dist} + CKu_{dem})$	$\uparrow 8(NCu_{dist} + CKu_{dem})$	$\uparrow 8(NC + 2CK + 4NK)$	$\uparrow 8(NCu_{dist} + CKu_{dem})$
add constraints	$S_{tot} \uparrow$ if collecting new data, else -	C_{comp}	$\downarrow$ if R decreases, else $\uparrow$	-	$\downarrow$ if only tries feasible solutions, else $\uparrow$	$\uparrow$	-
	$S_{req} \uparrow$	T_{comp}	$\uparrow$ extra constraints check	$\uparrow$ extra constraints check	$\uparrow$	$\uparrow$	$\uparrow$ extra constraints check
		C_{disk}	as C_{comp}	-	as C_{comp}	$\uparrow$	-
p.d. group 1 $\rightarrow$ 2 $\rightarrow$ 3	$S_{tot} \uparrow : S_{req} \uparrow$	C_{comp}	$\uparrow CNR \rightarrow CNRK \rightarrow CNRK$	$\uparrow CN \rightarrow CNK \rightarrow CNK$	$\uparrow I_{2dist}CN \rightarrow I_{2dist}CNK \rightarrow I_{2dist}CNK$	$\uparrow$	$\uparrow CN \rightarrow CNK \rightarrow CNK$
	$T_{comp} \uparrow$	C_{disk}	-	-	-	$\uparrow$	-
no $sc_{jk} \rightarrow sc_{jk}$	C_{comp} - : $T_{comp} \uparrow$: C_{disk} - : $S_{tot} \uparrow +8NK$ $S_{req} \uparrow +8N$ for single obj, $+8NK$ for mult. obj						

Table 4: General overall rules for how the decision time components change with problem definition.

in question increases, a $\downarrow$ that it decreases, and a "-" that there is no or little effect on the value of the parameter. The problem definition can be varied along four axes: the cost function can take into account single or multiple objects; constraints can be added or removed; the cost function can be taken either from group 1, 2 or 3 (see Table 1); finally, the cost function can consider storage cost or not. C_{msg} is not included, as it is always $N - 1$. The "For All" column lists the parameters that are independent of the approximation method. The "Parameter" column to the right of that lists the parameters that are approximation method dependent. The dependencies are then listed to the right of the parameter column for each approximation method.

For approximation methods Re(Lin), DP and PP, we were not able to give any general rules as heuristics within these groups might differ substantially. When going from group 1 to 2 to 3, S_{tot} scales as $NCu_{dist} \rightarrow NCu_{dist} + CKu_{dem} \rightarrow NCu_{dist} + 2CKu_{dem}$ and S_{req} as $NCu_{dist} \rightarrow NCu_{dist} + Cu_{dem} \rightarrow NCu_{dist} + 2Cu_{dem}$.