Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 68 (2008) 113–136
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Comparison and analysis of ten static heuristics-based Internet data replication techniques

Samee Ullah Khan, Ishfaq Ahmad*

Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, USA

Received 13 October 2004; received in revised form 12 February 2007; accepted 18 June 2007
Available online 19 July 2007

Abstract

This paper compares and analyzes 10 heuristics to solve the fine-grained data replication problem over the Internet. In fine-grained replication, frequently accessed data objects (as opposed to the entire website contents) are replicated onto a set of selected sites so as to minimize the average access time perceived by the end users. The paper presents a unified cost model that captures the minimization of the total object transfer cost in the system, which in turn leads to effective utilization of storage space, replica consistency, fault-tolerance, and load-balancing. The set of heuristics include six A-Star based algorithms, two bin packing algorithms, one greedy and one genetic algorithm. The heuristics are extensively simulated and compared using an experimental test-bed that closely mimics the Internet infrastructure and user access patterns. GT-ITM and Inet topology generators are used to obtain 80 well-defined network topologies based on flat, link distance, power-law and hierarchical transit–stub models. The user access patterns are derived from real access logs collected at the websites of Soccer World Cup 1998 and NASA Kennedy Space Center. The heuristics are evaluated by analyzing the communication cost incurred due to object transfers under the variance of server capacity, object size, read access, write access, number of objects and sites. The main benefit of this study is to facilitate readers with the choice of algorithms that guarantee fast or optimal or both types of solutions. This allows the selection of a particular algorithm to be used in a given scenario.

Keywords: Internet; Data replication; Static allocation; Informed searching

1. Introduction

With the exponential growth of the World Wide Web, popular web-servers are required to handle enormously large number of requests from a diverse range of users [1]. A major requirement of these web-servers is to provide better Internet usage, such as faster access time. The Internet access time simply cannot be improved by using high performance web-servers [51]. Efficient and sophisticated caching and replication techniques are necessary to ensure up-to-date contents, fast information retrieval, reduced web-server load, and added reliability.

Caching was traditionally applied to distributed file systems, such as the AFS [42]. Caching attempts to store the most commonly accessed objects as close to the clients as possible, while replication distributes a site's contents across multiple mirror servers. Although caching is a well-studied problem, its application on the Internet gave rise to new problems, e.g., where to place a cache, how to ensure the validity of cached contents, and how to handle dynamic pages. Replication, in contrast, has been commonly used in distributed systems to increase availability and fault-tolerance, as demonstrated in a real world example of Ocean Store [52]; side effects of replication are load-balancing and enhanced client–server proximity [53]. In general, replication and caching play complementary roles in the Internet environment [41]. Caching can be viewed as a special case of replication when mirror servers store only parts of a site's contents [1]. This analogy leads to some interesting comparisons. For instance, cache replacement algorithms are examples of on-line, distributed, locally greedy algorithms for data allocation in replicated systems. Furthermore, caches do not have full server capabilities and thus can be viewed as a replicated system that sends requests for specific object types (e.g., dynamic pages) to a single server. Essentially, every major

* Corresponding author. Fax: +1 817 272 3784.
E-mail addresses: sakhan@cse.uta.edu (S.U. Khan), iahmad@cse.uta.edu (I. Ahmad).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.06.009

aspect of a caching scheme has its equivalent in replicated systems, but not vice versa.

Replication can be coarse-grained (replication of an entire site or server) or fine-grained (replication of individual data items or objects). We detail these two popular problem formulations as follows.

Coarse-grained replication: This deals with replicating the entire website, which is similar to the celebrated distributed file allocation problem [54]. We can formally state the problem as: "*Choose M replicas among N potential sites* ($N > M$) *such that certain constraints are optimized*." These constraints could be memory, reduction in latency, communication cost, etc. Since the entire site is copied to the replication location, it is coarse-grained replication [42]. A majority of the initial work focused on coarse-grained replication. For example, the work proposed in [38] modeled the Internet as a tree. The work is also based on the assumption that the access requests to the proxies by the clients (which reside on the leaves of the tree) are always on the direct path(s) toward the servers. The work reported in [38] deals with the proxy server placement. The heuristic approach proposed in [25] solves the replication problem without requiring the full knowledge of the network topology, which is commonly assumed in many approaches [42]. The heuristics approach was further improved in [26] by introducing a refined constrained based policy. A comprehensive comparison of replication algorithms using realistic data from Internet log files is done in [50], where the Greedy approach [50], outperforms the Dynamic Programming approach [38], Randomized [50], and Hot Spot [50]. Although replicating the entire contents of the website can reduce the hit-miss ratio by considerable amounts, the coarse-grained replication model is rather simplistic approach [42].

Fine-grained replication model: This allows the replication of individual objects as apposed to the entire site. This approach has many advantages [51], such as it saves the server memory capacity, moves only those objects that are actually required to be reallocated, and provides load-balancing. The generalized fine-grained replication is known to be NP-complete not only for general graphs [41], but also for partitioned graphs [30]. The work proposed in [41] analyzed both static (such as, a modified Greedy based approach [44] and an Evolutionary method based on Genetic algorithms [23]) and adaptive (such as, a self-configured Genetic approach) replication techniques. In conclusion, the static Genetic approach outperformed others. The work was further extended [40] with comparisons to Linear Programming [5] and Linear Integer Programming [19]. In [55], the authors compared a localized Greedy, DEJAVU and a genetic algorithm, and found that the genetic algorithm outperformed both the heuristics, supporting the results reported in [40].

This paper focuses on fine-grained replication algorithms. The algorithms determine where and how many replicas to be placed, so as to maximize the system performance. The decision where to place the replicated data must trade off the cost of accessing the data, which is reduced by having additional copies, against the cost of storing and updating the copies. In general, clients experience reduced access latencies provided that the data are replicated within their close proximity. However, rapid updates (or writes) may counteract the replication benefit because of the overhead in maintaining a large number of replicas [41]. With reads and updates, the locations of the replicas have to be: (1) in close proximity to the client(s), and (2) in close proximity to the primary (assuming a broadcast update model) copy [33]. Therefore, the efficiency of a replication scheme depends strongly on the number of replicas and the selection of the placement sites [42]. Myriad theoretical approaches have been proposed that we can roughly classify into the following six categories:

Facility location: The facility location problem can be defined as follows: "*Find a location that minimizes a weighted sum of distances to each of several locations*." The generalized facility location problem is NP-complete [18]. The only known work on data replication with similar characteristics as that of the facility location problem is reported in [25]. However, the techniques reported are very tedious, have superfluous assumptions, and do not fully capture the concept of replicating a single item (object or site) over a fixed number of hosts [40].

File allocation: File allocation has been a popular line of research in distributed computing [14,44], distributed databases [2], multimedia databases [54], paging algorithms [16], and video server systems [54]. The generalized file allocation problem for multiple objects [11] has been proven to be NP-complete [15]. We can formally state the file allocation problem with context to the replication problem [42] as: "*For a network of M sites each with different storage capacity, replicate N files such that it satisfies the storage constraint and also optimizes some performance parameters e.g. network flow and/or reduce download speed*." File allocation has also been studied in the un-capacitated version [43]. The authors provided a guaranteed optimal result, but due to the assumption of unlimited capacity, the result is of little practical use [31].

Minimum k-median: The celebrated NP-complete minimum k-median problem is formally defined as follows [42]: "*Given is a graph* $G(V, E)$ *with weights on the nodes representing the number of requests and lengths on the edges. Satisfy a request, such that it minimizes the network cost of traversal and the path from the origin node and a server*." A lot of work has been done on k-median and its variants, e.g., [24,34]. In [38] the authors studied the problem of placing M proxies at N nodes where the topology of the network is a tree and proposed an $O(N^3M^2)$ algorithm. The result was further refined in [56]. Both the results carry theoretical significance but are impractical since the underlying topology was assumed to be a tree or requires the accesses to data be made on a well-defined minimum spanning tree residing inside the graph. A similar result with the objectives of minimizing the overall access cost by clients to the web sites and minimizing the longest delay can be found in [27]. To compliment these approaches, a more generalized solution was presented in [50], wherein the authors compared various placement techniques and proposed a greedy algorithm that outperformed other techniques including the work reported in [38]. Most of the results in this category have already been discussed while describing the coarse-grained model.

Capacity-constrained optimization: Constraint optimization is a class of problems that is widely studied in Operations Research. In the context of the data replication problem, the capacity-free (unlimited storage) version has a better worst-case performance than the capacity-constrained version [42], yet it requires a lot of maneuverability in terms of choosing the optimization function [9]. In [30], the authors use the capacity-constrained version of the minimum k-median problem and guarantee a stable performance. However, such results are possible only with highly conservative assumptions (such as, fixed location of the original server, access patterns are to be known beforehand, no network failures, etc.) as addressed in [26,37]. Therefore, they are not able to capture the dynamic nature of the system [42].

Bin packing: Widely studied in the field of on-line algorithms, the bin packing problem is known to be NP-hard [18]. We can formally state the problem with context to replication as [42]: "*Given N various objects of different sizes*, *partition them into the minimum number of disjoint sets such that the cumulative size of each set does not exceed a certain threshold*." This approach was first studied in [46], wherein the authors formulated the problem over a cluster of web-servers and reduced the server loads by using dummy replicas. However, their approach performs well only for small networks. Also of interest may be the more flexible and general approach undertaken in [32], which employed extensive experimental comparisons using various network topologies and real access logs.

Knapsack: To reduce network latency, a proactive web-server can decide where to place the copies of the objects in a distributed web-server, by employing partial replication [4]. Many researchers [46] have used the partial replication technique with the support of content-aware distributors. The primary usage of content-aware distributors is to redirect the client's request to the server that has the copy of the document requested [48]. In all of the above approaches the authors have primarily adopted the knapsack problem approach, which can be stated as follows: "*Given is a network of M nodes with distinguishable capacities and N objects. Find a subset of objects whose total size is bounded by the capacities for a site, and the total profit is maximized*." Here the profit is to reduce the communication cost or latency, etc. If no such replica is unassigned the problem is reduced to 0–1 knapsack [18]. Due to the close resemblance of the knapsack problem to the bin packing problem, it is widely studied by researchers in the field of Operations Research [58], Game Theory [20], and Approximation Algorithms [61]. It is proved that minimizing the maximum load over all the web-server nodes is NP-complete [10]. If the constraint of load-balancing is removed, the problem of minimizing the communication cost still remains NP-hard [60]. A rather different approach [12] using the concept of read-one-write-all policy has also been investigated in the context of dynamic data replication within the scope of 0–1 knapsack formulation. Some additional work of relevance is reported in [7,41,53]. A number of bibliographies and surveys for web caching are also available online, e.g., [13]. An overview of replication and its challenges are provided in [42] and [51], respectively.

This paper compares and analyzes 10 static heuristics-based techniques that address the problem of web content replication as a fine-grained (object based) replication. To provide a fair comparison, the assumptions and the system parameters are kept the same in all the approaches. The techniques studied in this paper are: bin packing heuristics using local (LMM) and global (GMM) knowledge, A-Star (DRPA-Star) and A-Star based heuristics (SA1, SA2, SA3, WA-Star, and A_ε-Star). For fine-grained replication, the algorithms proposed in [38,41,50] are the only ones that address the problem domain similar to ours. We select from [50] the greedy approach (Greedy) for comparison because it is shown to be the best compared with four other approaches (including the proposed technique in [38]); thus, we indirectly compare with four additional approaches as well. From [41], we select the genetic algorithm based replica placement technique (GRA). GRA is compared with a self-evolving Genetic algorithm (proposed by the same authors) and a modified Greedy based approach (originally proposed in [44]); thus, we indirectly compare with two additional techniques. Experimental results revealed that GRA outperformed the other techniques, and showed stability under various (system) environments [42]. We summarize the studied heuristics in Table 1. Except for Aε-Star and Wε-Star, all of the other techniques have been chosen from previous studies. Note that seven out of the eight previously reported techniques studied in this paper were in some shape and form proposed by the authors of this paper.

The main purpose of this paper is to study and compare the above mentioned heuristics on a unifying platform with varying system parameters so as to fully understand the capabilities and limitations of the methods. Selecting the best heuristic to be used in a given environment, however, remains a difficult task, since comparisons are often clouded by different underlying assumptions in the original study of the heuristic. This phenomenon can be observed by inspecting Table 2, where each heuristic is tested using various combinations of the input data and the underlying problem domain. For this purpose, a collection of 10 heuristics from the literature has been selected. This is certainly not an exhaustive selection of heuristics, nor it is meant to be. These 10 heuristics were selected since they seemed to be the most appropriate for the fine-grained data replication problem and covered a wide range of techniques, such as, informed searching, iterative, non-iterative, greedy, and naturally inspired techniques.

For a thorough testing of the heuristics, we make use of an experimental test-bed that closely mimics the Internet in its infrastructure and user access patterns. GT-ITM [6,59] and Inet [8] topology generators are used to obtain 80 well-defined network topologies based on flat, link distance, power-law and hierarchical transit–stub models. The user access patterns are derived from real access logs collected at the Soccer World Cup 1998 [3] web-server and NASA Kennedy Space Center web-server [47]. The idea of using access logs was first conceived and implemented in [50]. Although these access logs provide an accurate measure of the system load, yet they lack in capturing the combined temporal and spatial affects. This study leverages upon the original work reported in [50] to detail the

Table 1
Summary of the replica placement techniques studied in this paper

Method acronym	Method full name	Algorithmic category	Replica placement category	Original work	Original reporting work
DRPA-Star	Data replication method A-Star	A-Star	Capacity-constrained optimization	x	[32]
SA1	Suboptimal A-Star method 1	A-Star	Capacity-constrained optimization	x	[32]
SA2	Suboptimal A-Star method 2	A-Star	Capacity-constrained optimization	x	[32]
SA3	Suboptimal A-Star method 3	A-Star	Capacity-constrained optimization	x	[32]
WA-Star	Weighted A-Star method	A-Star	Capacity-constrained optimization	√	This paper
A_ε-Star	A-Star in range of ε	A-Star	Capacity-constrained optimization	√	This paper
LMM	Local Min–Min	Bin packing	Bin packing	x	[32]
GMM	Global Min–Min	Bin packing	Bin packing	x	[32]
Greedy	Greedy	Greedy algorithms	Capacity-constrained optimization	x	[50]
GRA	Genetic replica allocation method	Genetic algorithms	Knapsack	x	[41]

Table 2
The major work reported in the field of replica placements

Category	Work	Problem definition related assumptions					Experimental related information			
		Topology assumed	Access		Objects	Storage	Evaluated	Access patterns	Topologies	Comparisons
			Reads	Writes	Multiple/single	Constraints				
1	[25]	General graphs	Yes	No	Single	No	Yes	Synthetic (Zipf)	AS, transit–stub	None
2	[2]	General, fork graphs	Yes	Yes	Single	No	No	N/A	N/A	None
	[11]	Fully connected graphs	Yes	Yes	Single	No	No	N/A	N/A	None
	[15]	General graphs	Yes	No	Single	No	Yes	Statistical	Flat	None
	[16]	Uniform cost graphs	Yes	No	Multiple	No	No	N/A	N/A	None
	[36]	General graphs	Yes	Yes	Multiple	Yes	Yes	Statistical	Flat	None
	[44]	General graphs	Yes	Yes	Single	No	No	N/A	N/A	None
	[54]	General graphs	Yes	Yes	Multiple	Yes	Yes	Statistical	Flat	[36]
3	[24]	Linear, general graphs	Yes	No	Single	No	No	N/A	N/A	[34]
	[30]	General graphs	Yes	No	Multiple	No	Yes	Synthetic (Zipf)	AS	[25,28]
	[27]	Trees	Yes	No	Single	No	Yes	Statistical	Trees	[38]
	[34]	General graphs	Yes	No	Multiple	Yes	No	N/A	N/A	None
	[38]	Trees	Yes	No	Single	No	Yes	Statistical	Trees	None
	[56]	Trees, general graphs	Yes	No	Multiple	No	Yes	Statistical	Tress, Flat	[38]
4	[9]	Linear, general graphs	Yes	No	Multiple	No	No	N/A	N/A	None
	[26]	General graphs	Yes	Yes	Multiple	No	Yes	Synthetic (Zipf)	AS, transit–stub	[25]
	[37]	General graphs	Yes	Yes	Multiple	Yes	Yes	Real-time	Flat	None
	[50]	**Trees, general graphs**	**Yes**	**Yes**	**Multiple**	**Yes**	**Yes**	**Access logs**	**Tress, flat**	**[38]**
5	[32]	**General graphs**	**Yes**	**Yes**	**Multiple**	**Yes**	**Yes**	**Access logs**	**Flat**	**[41,50]**
	[46]	Linear graphs	Yes	Yes	Multiple	Yes	Yes	Statistical	Flat	None
6	[7]	General graphs	Yes	Yes	Single	No	No	N/A	N/A	None
	[12]	Linear, general graphs	Yes	Yes	Single	No	No	N/A	N/A	None
	[41]	**General graphs**	**Yes**	**Yes**	**Multiple**	**Yes**	**Yes**	**Synthetic (Zipf)**	**Flat**	**[44]**
	[53]	General graphs	Yes	Yes	Multiple	No	No	N/A	N/A	None

Works represented in bold will be compared and analyzed in this paper.

performance of the heuristics using both spatial, the Soccer World Cup 1998 access logs, and temporal, NASA Kennedy Space Center access logs. Using our experimental setup, the heuristics are evaluated by analyzing the system utilization in terms of reducing the communication cost incurred due to object transfer(s) under the variance of server capacity, object size, read access, write access, number of objects and sites. Based on the experimental data, we were able to comparatively examine the behavior of each of the techniques.

For the replica placement heuristics studied here, the solution qualities varied considerably with each system parameter alteration. For instance:

1. By varying system storage capacity, the heuristics projected a projected increase in the solution quality. However, this was true only when accesses to objects had a temporal rapport, when subjected to the spatially skewed NASA Kennedy Space Center access log, the solution qualities

of the studied heuristics showed abrupt changes without any correlation to the constantly increasing system storage capacity. Under such conditions (we observed that the), heuristics that construct solution(s) incrementally (such as, Greedy, LMM, and GMM) performed relatively better than the heuristics that developed combinatorial (all other studied techniques) solutions. This is because the heuristics that built incremental solution(s) focused on hot-spots [51] before considering the rest of the network. (Dealing with hot-spots first, would bring in more benefit than otherwise.) A spatially skewed access log is nothing more than a hot-spot concentrated access log.

2. Varying read/write access frequencies also brought out a unique facet of the comparative analysis. Usually, an increase in the read frequency would advocate that the system should be populated with as much replicas as possible. However, this may not be a clever solution, since it will increase the cost of replica maintenance. On the other hand, an increase in the update frequency would tend to reduce the number of replicas. This, also, may not be a viable strategy since it may increase the cost of accessing objects by considerable amounts. Among the studied heuristics, the class of heuristics that constructed combinatorial solution(s) adapted well to the increase and decrease in the read/write frequencies compared to the heuristics that developed incremental solution(s), with one exception—the Greedy technique. It increased its solution quality when the system was subjected to the spatially skewed NASA Kennedy Space Center access log, and maintained a fair performance with the Soccer World Cup 1998 access logs.
3. By keeping the read/write access frequencies and the system storage capacity fixed, the mere increase in the number of sites or objects severely alters the performances of the heuristics. For instance, the increase in the number of sites in the network has a dual effect: (1) it increases the system storage capacity; (2) it increases the hot-spot phenomenon. We observed some very interesting results in this respect. LMM and GMM being bin packing based heuristics were quick to exploit the increase in the system storage capacity aspect, but failed to maintain their supremacy with continued increase in the hot-spots (number of sites). This is because initially the combinations of bins (sites) increased, but with the further increase in the number of bins, the effect was not as observable as all the essential objects were already packed (replicated). Some heuristics, for instance, WA-Star and Aε-Star showed a constant performance. They seemed to adapt rapidly to the bi-polar property, because of their combinatorial solution construction. Other A-Star based heuristics, namely, SA1, SA2, and SA3 also construct combinatorial solutions; however, their performance was not very impressive. This is because WA-Star and Aε-Star operate on altering the priority of the search paths, while SA1, SA2, and SA3 simply truncate search paths. Thus, when the number of sites increase, the problem size increases, and so does the search tree. Truncating search paths would have a severer impact on the solution quality compared to altering the search paths.
4. Taking into consideration the problem size alone, A-Star and A-Star based heuristics performed extremely well when the problem size was small. This is because these heuristics rely on exhaustive (A-Star) or sub-exhaustive (A-Star based heuristics) search patterns. Intuitively, the A-Star technique being reliant on an exhaustive search pattern, failed to identify any solution for a problem size that contained more than 60 sites and 300 objects. The genetic algorithm also performed well, since under small problem instances, the (gene) population to solution candidate (next generation) ratio is extremely high. With larger problem size, this ratio reduces and so does the solution quality. The A-Star based heuristics, which were designed to speedup the search patterns finished in some cases faster than the genetic algorithm, specifically SA2 and Aε-Star, but with a superior solution quality.

It can be gathered from the above discussion that the problem at hand has certain very important parameters, such as, system storage capacity, read write ratio, number of sites and objects in the system, etc., that affect the performance of a replica placement technique by considerable amounts. One cannot (with confidence) say that a particular replica placement technique would be sufficient for each and every scenario. This study is an effort to provide a benchmark by identifying scenarios and comparing heuristics on a unifying platform with changing system parameters for the convenience of network designers, Internet operators, researchers, etc. Based on our observations, we also make suggestive uses of each and every heuristic and identify the circumstances in which they are deemed useful.

The remainder of this paper is organized as follows. In Section 2 we underlie the system model and the necessary system related assumptions. Section 3 describes the data replication problem. Section 4 describes the placement policies used in the paper. The experimental results and concluding remarks are provided in Sections 5 and 6, respectively.

2. System model and assumptions

The system model under consideration is a large-scale distributed computing system, where users access data objects which are held by the sites. We elucidate the few system related assumptions as follows.

1. Each site is assigned a unique site identifier. There a total of M sites in the system and S^i $(1 \leqslant i \leqslant M)$ denotes a site identifier.
2. Each data object is assigned a unique object identifier. There a total of N data objects in the system and O_k $(1 \leqslant k \leqslant N)$ denotes a data object identifier.
3. The original copy of an object is held by a particular site in the system called the primary site, denoted as P_k. This site also holds the information about where the replicas of object O_k reside in the system.
4. Each site has a limited storage capacity which is denoted by s^i.
5. The read and write access frequencies are known a priori (or observed through access log).

6. For updates we assume a "broadcast" or lazy replication model [33]. In this model when an object is updated, the update is sent to the site (P_k) which holds the original copy of the object. P_k upon receiving the updated contents broadcasts the updates to the sites which hold the replicas of the object. In this way, we can always guarantee that the data contents in the system are up-to-date.

Based on the above system overview and the underlying assumptions, if we are to find an optimal placement of replicas in a large-scale distributed computing system, then we must incorporate among others the following parameters in a brute force (exhaustive search) method [40]:

1. The access frequency of each data object.
2. The time remaining until each data object is updated next.
3. The probability that each site functions properly during the lifespan of the system.
4. The probability that the network will remain connected during the lifespan of the system.

Even if some lopping is possible, the computational complexity is very high, and this calculation must be done every time any of the above parameters change. Moreover, parameters 3 and 4 cannot be formulated in practice because faults do not follow a known phenomenon. For these reasons, we take the following heuristic approach:

1. Replicas are relocated in a specific period, the *relocation period*. The term relocation period was first coined in [22]. There the author proposed several very effective replica placement techniques for the ad hoc network environment. Since dynamics vary significantly in an ad hoc environment, replica placements in [22] were made during a specific time frame called the relocation period.
2. At every relocation period, replica allocation is determined based on the access (both read and update) frequency of each data object and the network topology at that moment.

3. The data replication problem description

The most frequently used acronyms in this paper are listed in Table 3.

Table 3
Notations and their meanings

Symbols	Meaning
M	Total number of sites in the network
N	Total number of objects to be replicated
O_k	kth object
o_k	Size of object k
S^i	ith site
s^i	Size of site i.
r_k^i	Number of reads for object k from site i
R_k^i	Aggregate read cost of r_k^i
w_k^i	Number of writes for object k from site i
W_k^i	Aggregate write cost of w_k^i
NN_k^i	Nearest neighbor of site i holding object k
$c(i, j)$	Communication cost between sites i and j
P_k	Primary site of the kth object
R_k	Replication schema of object k
C_{overall}	Total overall data transfer cost
DRP	Data replication problem
RC	Replication cost (network communication cost)
W-log	Soccer World Cup 1998 access log
N-log	NASA Kennedy Space Center access log

Consider a distributed system comprising M sites, with each site having its own processing power, memory (primary storage), and media (secondary storage). Let S^i and s^i be the name and the total storage capacity (in simple data units e.g. blocks), respectively, of site i where $1 \leqslant i \leqslant M$. The M sites of the system are connected by a communication network. A link between two sites S^i and S^j (if it exists) has a positive integer $c(i, j)$ associated with it, giving the communication cost for transferring a data unit between sites S^i and S^j. If the two sites are not directly connected by a communication link then the above cost is given by the sum of the costs of all the links in a chosen path from site S^i to the site S^j. Without the loss of generality we assume that $c(i, j) = c(j, i)$. This is a common assumption e.g. see [25,32,41,50]. Let there be N objects, each identifiable by a unique name O_k and size in simple data unites o_k where $1 \leqslant k \leqslant N$. Let r_k^i and w_k^i be the total number of reads and writes, respectively, initiated from S^i for O_k during a certain time period t. This time period t determines when to instigate the relocation period so that the replica placement algorithm can be invoked. Note that this time period t is the only parameter that requires human intervention. However, in this paper we use analytical data that will enable us to effectively predict the time interval t (see Section 5.4 for details).

Our replication policy assumes the existence of one primary copy for each object in the network. Let P_k, be the site which holds the primary copy of O_k, i.e., the only copy in the network that cannot be de-allocated, hence referred to as primary site of the kth object. Each primary site P_k, contains information about the whole replication scheme R_k of O_k. This can be done by maintaining a list of the sites where the kth object is replicated at, called from now on the *replicators* of O_k. Moreover, every site S^i stores a two-field record for each object. The first field is its primary site P_k and the second the nearest neighborhood site NN_k^i of site S^i which holds a replica of object k. In other words, NN_k^i is the site for which the reads from S^i for O_k, if served there, would incur the minimum possible communication cost. It is possible that $NN_k^i = S^i$, if $S^i$is a *replicator* or the primary site of O_k. Another possibility is that $NN_k^i = P_k$, if the primary site is the closest one holding a replica of O_k. When a site S^i reads an object, it does so by addressing the request to the corresponding NN_k^i. For the updates we assume that every site can update every object. Updates of an object O_k are performed by sending the updated version to its primary site P_k, which afterwards broadcasts it to every site in its replication scheme R_k.

For the DRP under consideration, we are interested in minimizing the total replication cost (RC) or the total network transfer cost. The communication cost of the control messages has minor impact to the overall performance of the system [42], therefore, we do not consider it in the transfer cost model, but it is to be noted that incorporation of such a cost would be a

trivial exercise. There are two components affecting RC. The first component of RC is due to the read requests. Let R_k^i denote the total RC, due to S^is' reading requests for object O_k, addressed to the nearest site NN_k^i. This cost is given by the following equation:

$$R_k^i = r_k^i o_k c(i, NN_k^i), \tag{1}$$

where $NN_k^i = \{Site j | j \in R_k^{\wedge} \min c(i, j)\}$. The second component of RC is the cost arising due to the writes. Let W_k^i be the total RC, due to S^is' writing requests for object O_k, addressed to the primary site P_k. This cost is given by the following equation:

$$W_k^i = w_k^i o_k \left(c(i, P_k) + \sum_{\forall (j \in R_k), j \neq i} c(P_k, j) \right). \tag{2}$$

Here, we made the indirect assumption that in order to perform a write we need to ship the whole updated version of the object. This of course is not always the case, as we can move only the updated parts of it (modeling such policies can also be done using our framework). The cumulative RC, denoted as C_{overall}, due to reads and writes is given by

$$C_{\text{overall}} = \sum_{i=1}^{M} \sum_{k=1}^{N} (R_k^i + W_k^i). \tag{3}$$

Let $X_{ik} = 1$ if S^i holds a replica of object O_k, and 0 otherwise. X_{ik}s define an $M \times N$ replication matrix, named X, with Boolean elements. Eq. (3) is now refined to

$$X = \sum_{i=1}^{M} \sum_{k=1}^{N} (1 - X_{ik})[r_k^i o_k \min\{c(i, j) | X_{jk} = 1\} + w_k^i o_k c(i, P_k)] + X_{ik} \left(\sum_{x=1}^{M} w_k^x \right) o_k c(i, P_k). \tag{4}$$

Sites which are not the *replicators* of object O_k create RC equal to the communication cost of their reads from the nearest *replicator*, plus that of sending their writes to the primary site of O_k. Sites belonging to the replication scheme of O_k, are associated with the cost of sending/receiving all the updated versions of it. Using the above formulation, the DRP can be defined as

"Find the assignment of 0, 1 values in the X matrix that minimizes C_{overall}, *subject to the storage capacity constraint*:

$$\sum_{k=1}^{N} X_{ik} o_k \leqslant s^i \forall (1 \leqslant i \leqslant M),$$

and subject to the primary copies policy:

$$X_{P_k k} = 1 \quad \forall (1 \leqslant k \leqslant N)."$$

The minimization of C_{overall} has the following two impacts on the distributed computing system under consideration. First, it ensures that the object replication is done in such a way that it minimizes the maximum distance between the replicas and their respective primary objects. Second, it ensures that the maximum distance between an object k and the user(s) accessing that object is also minimized. Thus, the solution aims for reducing the overall RC of the system. In the generalized case, the DRP is proven to be NP-complete [41].

4. The replica placement techniques

4.1. A-Star based technique (DRPA-Star)

A-Star is a best-first search algorithm based on a μ-ary tree [49]. It starts from the root, called the start node (usually a null solution of the problem). Intermediate tree nodes represent the partial solutions, and leaf nodes represent the complete solutions or goals. A cost function f computes each node's associated cost. The value of f for a node n, which is the estimated cost of the cheapest solution through n, is computed as: $f(n) = g(n) + h(n)$, where $g(n)$ is the search-path cost from the start node to the current node n and $h(n)$, called the *heuristic*, is a lower-bound estimate of the path cost from n to the goal node (solution). The A-Star based searching technique for the data replication problem (DRPA-Star) starts from an assignment P, and explores all the *potential* options of assigning an object to a site. With proper pruning techniques used against the constraint(s) C, only the assignments in the admissible head set are explored. If the new solution is consistence with the constraint, it is added to the Expansion Tree (ET), otherwise the solution is pruned. In order to avoid memory overflow, we limit the ET to 1000 active solution (state) space allocations. This is a very common technique used for memory bounded A-Star type algorithms (for further details on memory bounded A-Star techniques see [29]). Moreover, the candidate objects assignments are ordered (in a linked list termed as the OPEN list), such that the smallest projected cost of allocation is expanded first. Thus, we can terminate our expansion when the solution for DRP is obtained, or there are no more candidate allocations left in the ET. In either case optimality is always guaranteed. DRPA-Star uses the following heuristic:

Let O_k and S^i represent the set of objects and sites in the system. Let U be the set of unassigned objects and t be the global minimum of an object's RC. Thus, we can define the minimum of such a cost as a set: $T = \min_{0 \leqslant j \leqslant N-1} (t(O_k, S^i)), \forall O_{k \in} U$. For a node n, let $mmk(n)$ define the maximum element of set T (the max–min RC). $mmk(n)$ then represents the best possible replica allocation without the unrealistic assumption that every object in U can be replicated to a site in M without a conflict. The heuristic used thus becomes: $h(n) = \max(0, [mmk(n) - g(n)])$, where $g(n)$ is equivalent to the cost of replicating an object onto a site, i.e., $g(n)$ is equivalent to the RC of object O_k onto site S^i. Pseudo-code for DRPA-Star is shown in Fig. 1.

Lemma 1. *DRPA-Star always identifies a solution, if there exists one.*

Proof. DRPA-Star expands its solution set by choosing the head of the OPEN list in the increasing order of the projected

```
DRPA-Star Algorithm
Inputs:
C_i (Replication cost matrix)
s_i (Array storing size of objects)
S^k (Array storing size of sites)
Output:
Final allocation of replicas
Initialize:
OPEN = NULL
Sol = NULL
Solution = False
Compute:
1. Create Start node s                                   /* Initialize the μ-ary tree */
2. Insert s into OPEN
3. while(OPEN != NULL or Solution = true)
4.      sort(OPEN)
5.      k ← Remove head of list*
6.      if k is the solution then        /* k can only be a solution when there is a 1-1 mapping between objects and sites*/
7.              Sol ← k ⊗ Sol
8.              Update storage constraints
9.              if no more replications possible          /* because of the storage constraints */
10.                     Solution = true
11.             endif
12.     endif
13.     Generate the successors of k                      /* Successors can only be generated when k is not an OAS */
14.     for every successor n' of k                       /* Construct the μ-ary tree */
15.             f(n') = g(n')+h(n')
16.     if n' satisfies storage constraint
17.             Insert n' into OPEN
18.     endif
19. endw
20. Output (Solution)

* k is also known as OAS (Objects Assigned to Sites).
```

Fig. 1. Pseudo-code for DRPA-Star.

cost. Since all the feasible candidates enter into the OPEN list, they must eventually expand the solution set to reach a feasible OAS solution. This would hold true if there are not infinitely many states with $h(x) = h(goal)$. In the DRP solution collection phase since every OAS is an optimal solution on the global constraint of storage capacity, an XORed solution of two consecutive OASs and potentially n OAS with cascaded XOR of assignments would eventually result in the solution for the DRP. □

Lemma 2. *DRPA-Star always chooses the best solution, if two or more solutions exist.*

Proof. Let u and v be two feasible allocations to OAS. Let c_u and c_v define the RC for u and v, respectively, and $c_u \leqslant c_v$. An optimal search will reach the solution u before it reaches v. Assume DRPA-Star identifies v prior to u. Thus, when DRPA-Star expands v, there would be some solution u not yet explored and that would imply $h_u \leqslant h_v$. Since by definition: $h_u = h_v$, the OPEN list ordering would ensure that the sorting is done according to the smallest expected cost from the current node to the solution, u would have been explored first. Thus, indeed DRPA-Star would reach v before u. Moreover, DRPA-Star will eventually identify the best global solution for DRP from the above arguments and Lemma 1. □

Lemma 3. *DRPA-Star grows and requires sub-exponential time and space, respectively.*

Proof. Let P be the expanded paths (partial or complete OAS solution) in the search tree, then the space required by the DRPA-Star is P and the time required by DRPA-Star is $dP(h + \log(P))$. Where d is the degree of the network, h is the depth at which the OASs are identified, and the $\log(P)$ factor identifies the growth of the search tree. Now the error in the heuristic grows no faster than log of the optimal cost of the solution. A-Star has been proven to be sub-exponential [49]. Since DRPA-Star due to its pruning is far more efficient than A-Star, DRPA-Star will also grow sub-optimally. We give the relation of sub-optimality as: $OPT_{\text{cost}} - A_{\text{cost}} = \text{O}(\log(OPT_{\text{cost}}))$, where A_{cost} is the admissible cost. We can thus say that $P \leqslant Mhd \leqslant dM^2$. For an average case analysis DRPA-Star uses space equivalent to Mhd, and thus the running time would be $Mhd^2(h + \log(Mhd))$. □

4.2. A-Star based refinements (WA-Star and Aε-Star)

Arguably DRPA-Star is an algorithm that goes about the optimization mission too seriously. Therefore, we are interested in identifying ways to quickly reach a solution though perhaps sub-optimally. One approach to address this predicament is to examine the effects of g and h separately. The effect of g is to add a breath-first component to the search. Without h, DRPA-Star would reduce to a pure breath-first search. On the other hand without g, DRPA-Star would ignore the distance already covered and would base its decision entirely on h, the estimate of the remaining proximity to the goal.

4.2.1. WA-Star

To cater for the two tendencies of A-Star type algorithms (as mentioned in Section 4.2), a weighted evaluation function is recommended: $f(n) = (1-w) \times g(n) + w \times h(n)$. Analytical results [49] have shown that w can have three values, i.e., 0, $\frac{1}{2}$, and 1 corresponding to exhaustive, A-Star, and breath-first search, respectively. Rather than keeping w constant throughout the search, it is natural to dynamically change w so as to weigh h less heavily as the search goes deeper. Thus, an effective evaluation function would be: $f(n) = g(n) + h(n) + \varepsilon[1 - (d(n)/D)]\ h(n)$, where $d(n)$ is the depth of node n and D is the anticipated depth of the desired goal node. It is to be noted that shallow levels of the search tree, i.e., when $d \ll D$, h is given a supportive weight equal to $1 + \varepsilon$, encouraging depth-first excursions. At deep levels, however, the search resumes an admissible equal weight, to avoid early termination. We call this variation of DRPA-Star as WA-Star.

Lemma 4. *WA-Star identifies a solution within a range of* $1+\varepsilon$ *of DRPA-Star.*

Proof. If $h(n)$ is admissible, then the algorithm is ε-admissible, that is, it finds a path from start to the goal node with a cost at most $1+\varepsilon$. This follows by observing that before the termination of the algorithms, the shallowest OPEN node n' along any optimal solution path has its cost ($g(n')$) equal to the optimal admissible cost ($g*(n')$) [49]. Therefore, we have

$$\begin{aligned} f(n') &\leqslant g^*(n') + h^*(n') + \varepsilon[1 - (d(n')/D)]h^*(n') \\ &\leqslant \varepsilon h^*(n') \\ &\leqslant 1 + \varepsilon. \qquad \square \end{aligned}$$

4.2.2. Aε-Star

Perhaps a natural way to speedup any searching technique is to focus on a solution space that somehow can guarantee that search in that particular space would not deviate from the optimal solution by a factor of, say ε. Keeping this in mind we propose an extension of the DRPA-Star technique, called Aε-Star. This technique uses two lists: OPEN and FOCAL. The FOCAL list is a sub-list of OPEN, and contains only those nodes that do not deviate from the lowest f node by a factor greater than $1 + \varepsilon$. That is, we can say that

$$\text{FOCAL} = \left\{ n \,|\, f(n) \leqslant (1+\varepsilon) \min_{n' \in \text{OPEN}} f(n') \right\}.$$

The technique works similar to DRPA-Star, with the exception that the node selection (lowest h) is done not from the OPEN but from the FOCAL list. The main intuition behind Aε-Star is that according to the estimates of f, all nodes in FOCAL have roughly equal solution paths. Therefore, rather than spending time on deciding which among them is the best, it makes more sense to use the time to compute the remaining portion of the solution from within FOCAL. (Notice that when $\varepsilon = 0$, Aε-Star reduces to DRPA-Star.) It is easy to see that this approach will never run into the problem of memory overflow. Moreover, the FOCAL list always ensures that only the candidate solutions within a bound of $1 + \varepsilon$ of DRPA-Star are expanded.

Lemma 5. *Aε-Star identifies a solution within a range of* $1+\epsilon$ *of DRPA-Star.*

Proof. Let n' be a node in OPEN list having the smallest f value, t be the termination node, n be the shallowest OPEN node on an optimal path, and $f(t)$ be the cost of the solution found. Then we can say that:

$$\begin{aligned} f(n') &\leqslant f(n) \ \text{(Since } h \text{ is admissible and OPEN is } f \text{ ordered)} \\ f(t) &\leqslant f(n')(1+\varepsilon) \quad \text{(Since } t \text{ is chosen from FOCAL.)} \\ &\leqslant f(n)(1+\varepsilon). \qquad \square \end{aligned}$$

4.3. DRPA-Star based heuristics (SA1, SA2, and SA3)

We now present three heuristics (suboptimal A-Star) algorithms, referred to hereafter as SA1, SA2, and SA3. The name SA comes from Suboptimal Assignments. The main purpose is to design algorithms that converge to solution faster and overcome the high memory requirements associated with A-Star type algorithms [26]. The basic idea in these algorithms is that when the search process reaches a certain depth in the search tree, some search path(s) can be avoided (some tree nodes can be discarded) without moving far from the optimal solution.

4.3.1. SA1

In SA1, when the algorithm (DRPA-Star) selects a node that belongs to level R or below, it generates only the best successors (lowest expansion cost) of it. All the other successors except the best one are discarded.

4.3.2. SA2

In SA2, when the depth level R is reached for the very first time, all the successors except the minimum cost are discarded among all the nodes marked for expansion.

4.3.3. SA3

In SA3, the discarding is done similar to SA2 except that now the nodes are removed from the ET. For instance, if n nodes are generated, then all of them are inserted in the ET, and the $n - 1$ high cost nodes are discarded.

These techniques will not suffer from memory overflow, since at level R, for every node taken out of the ET for expansion, only one node is inserted. Also the running time is reduced by many folds since the algorithm expands/explores less number of nodes when it reaches R.

4.4. Bin packing based techniques (LMM and GMM)

The bin packing problem formulation resembles the DRP in many ways; therefore, it is natural to see the DRP as a special case of the bin packing problem. In such a setup, the sites of the

distributed system can be considered as the bins with specified storage capacity, and the objects can be considered as the items that need to be packed in the bins such that the total storage capacity of the bins is not exceeded and the profit brought by packing more items inside the bins is maximized. Here the profit can be made equivalent to minimizing the RC cost.

4.4.1. Local Min–Min (LMM)

Let O_k and S^i represent the set of objects and sites in the system. Let U be the set of unassigned objects to a site S^i. Let $U_{\min}$ define the minimum RC of the objects to be assigned to a particular site. The assignment is made in the ascending order of set U. If there is a tie among two objects, then the tie is broken by the minimum object size, hence the name Min–Min. Since we do the assignment iteratively for every object and do not consider the effects of the choice of an object to a site with respect to other sites, we call it Local Min–Min (LMM).

Lemma 6 (*Khan and Ahmad [32]*). *LMM converges in* $O(MN(\log N))$ *and requires linear space*.

4.4.2. Global Min–Min (GMM)

Let O_k and S^i represent the set of objects and sites in the system. Let U be the set of unassigned objects and k be the global minimum of all the RC associated with an object. The minimum of such cost as a set $T = \min_{0 \leqslant j \leqslant N-1} (k(O_k, S^i), \forall O_{k \in} U$. If during the assignment, the minimum RC of an object is the same for two different sites, the object is chosen on random. For a node n let $mink(n)$ define the minimum element of set T. Thus $mink(n)$ represents the best minimum RC that would occur if object O_k is replicated to a site S^i, i.e., Global Min–Min (GMM).

Lemma 7 (*Khan and Ahmad [32]*). *GMM requires* $O(M^2N^2(\log N))$ *running time and* $\Omega(MN)$ *space*.

4.5. Greedy based technique (Greedy)

The Greedy algorithm reported in [50] works in an iterative fashion. In the first iteration, all the M sites are investigated to find the replica location(s) of the first among a total of N objects. Consider that we choose an object i for replication. The algorithm recursively makes calculations based on the assumption that all the users in the system request for object i. Thus, we have to pick a site that yields the lowest cost of replication for the object i. In the second iteration, the location for the second site is considered. Based on the choice of object i, the algorithm now would identify the second site for replication, which, in conjunction with the site already picked, yields the lowest RC. Observe here that this assignment may or may not be for the same object i. The algorithm progresses forward till either one of the DRP constraints are violated. Further details about the Greedy algorithm can be obtained from [50].

Lemma 8 (*Qiu et al. [50]*). *Greedy requires* $O(M^2 N)$ *running time*.

4.6. Genetic algorithm based technique (GRA)

In [41] the authors proposed a genetic algorithm based heuristic, called Genetic Replication Algorithm (GRA). GRA provides good solution quality, but suffers from slow termination time. This algorithm is chosen since it was the first work that realistically addressed the fine-grained replication on the same problem formulation as taken in this paper. The technique shows great stability under various scenarios which have been experimentally derived. Briefly, the GRA exploits the mix and match technique. Chromosomes represent the various replication schemas and each consists of M genes (one for each site). Every gene is composed of N bits (one for each object). A 1 value in the kth bit of the ith gene denotes that the ith site holds a replica of the kth object, and 0 otherwise. Using this chromosome encoding, crossover, mutation, and selection operations are performed to report the best chromosome as the final solution. Readers are encouraged to see [41] for further details about the GRA method.

Lemma 9 (*Loukopoulos and Ahmad [41]*). *GRA requires* $O(N_g N_p M^2 N + N_p M N^2)$ *running time*.

5. Experimental setups, results and discussion

We performed experiments on a 440 MHz Ultra 10 machine with 512 MB memory. The experimental evaluations were targeted to benchmark the placement policies. In all the experiments for the A-Star based suboptimal heuristics (SA1, SA2, and SA3), the cutoff R was set at: $R = \lfloor d/2 \rfloor$, where d represents the depth of the tree (number of objects). For WA-Star and Aε-Star, ε was set to be the mean value of g in the OPEN and FOCAL lists, respectively.

5.1. Performance metric

The solution quality in all cases, was measured according to terms of the RC percentage that was saved under the replication scheme found by the algorithms, compared to the initial one, i.e., when only primary copies exist.

5.2. Network topologies

To establish diversity in our experimental setups, the network connectivity was changed considerably. We used four types of network topologies, which we explain as follows. (All in all we used 80 various topologies.)

5.2.1. Flat methods

In flat random methods a graph $G = (V, E)$ is built by adding edges to a given set of nodes V subject to a probability function $P(u, v)$, where u and v are arbitrary nodes of G. We articulate the two flat methods used in this study as follows.

Pure random model: A random graph $G(M, P(\text{edge} = p))$ with $0 \leqslant p \leqslant 1$ contains all graphs with nodes (servers) M in which the edges are chosen independently and with a

probability p. Although this approach is extremely simple, yet it fails to capture significant properties of Web-like topologies [21]. The five pure random topologies were obtained using GT-ITM [59] topology generator with $p = \{0.4, 0.5, 0.6, 0.7, 0.8\}$.

Waxman model: The shortcomings of pure random topologies can be overcome by using the Waxman model. In this method edges are added between pairs of nodes (u, v) with probability $P(u, v)$ that depends on the distance $d(u, v)$ between u and v. The Waxman model is given by [57]

$$P(u, v) = \beta e^{\frac{-d(u,v)}{L\alpha}},$$

where L is the maximum distance between any two nodes and $\alpha, \beta \in (0, 1]$. β is used to control the density of the graph. The larger the value of β the denser is the graph. α is used to control the connectivity of the graph. The smaller the value of α the larger is the number of short edges [21]. The 12 Waxman topologies were obtained using the GT-ITM [59] topology generator with values of $\alpha = \{0.1, 0.15, 0.2, 0.25\}$ and $\beta = \{0.2, 0.3, 0.4\}$.

5.2.2. Link distance models

In pure random and Waxman models, there is no direct connection among the communication cost and the distance between two arbitrary nodes of the generated graph. To compliment these two models, we propose a class of graphs in which the distance between two nodes is directly proportional to the communication cost. In such methods, the distance between two servers is reverse mapped to the communication cost of transmitting a 1 kB of data, assuming that we are given the bandwidth. That is, the communication cost is equivalent to the sum of the transmission and propagation delay. The propagation speed on a link is assumed to be 2.8×10^8 m/s (copper wire). Thus, if we say that the distance between two nodes is 10 km and has a bandwidth of 1 Mbps, then it means that the cost of communication of 1 kB of data between the two nodes is equivalent to $10\,\text{km}/(2.8 \times 10^8\,\text{m/s}) + 1\,\text{kB}/(1\,\text{Mbps}) = 8.03$ ms, and the cost would simply be 0.00803. We detail the four link distance models used in this study as follows.

Random graphs: This method involves generating graphs with random: node degree (d^*), bandwidth (b), and link distance (d) between the nodes of the graph. We detail the steps involved in generating random graphs as follows. First, M (user input) nodes are placed in a plane, each with a unique identifier. Second, from the interval d^*, each node's out degree is generated. (At this moment the links do not have weights or communication costs.) Third, each link is assigned bandwidth (in Mbps) and distance (in km) on random. Finally, for each link the transmission and propagation delay is calculated, based on the assigned bandwidth and distance. The 12 random topologies were obtained using, $d^* = \{10, 15, 20\}$, $b = \{1, 10, 100\}$, and $d = \{5, 10, 15, 20\}$.

Fully connected random graphs: This method is similar to the technique described for the random graphs except that now we do not require the node degree since the entire graph is fully connected. The five random topologies were obtained using, $b = \{1, 10, 100\}$ and $d = \{d_1 = [1, 10], d_2 = [1, 20], d_3 = [1, 50], d_4 = [10, 20], d_5 = [20, 50]\}$. Notice that d has five elements $d_1, \ldots, d_5$. Each element was used to generate a particular graph. For instance, for the first graph, we choose the bandwidth randomly from the values of $\{1, 10, 100\}$, and the link distance randomly from the interval of $d_1 = [1, 10]$.

Fully connected uniform graphs: This method is similar to the fully connect random graphs technique except that the bandwidth and link distance are chosen uniformly and not randomly. The five random topologies were obtained using, $b = [1, 100]$ and $d = \{d_1 = [1, 10], d_2 = [1, 20], d_3 = [1, 50], d_4 = [10, 20], d_5 = [20, 50]\}$.

Fully connected lognormal graphs: This method is similar to the fully connect random graphs technique except that link distance is chosen log-normally and not randomly. Note that the bandwidth is still assigned on random. (Curious readers are encouraged to see [17] for an insight on the lognormal distribution functions.) The nine lognormal topologies were obtained using $b = \{1, 10, 100\}$ and $d = \{\mu = \{8.455, 9.345, 9.564\}, \sigma = \{1.278, 1.305, 1.378\}\}$, where μ and σ are the mean and variance parameters of the lognormal distribution function, respectively.

5.2.3. Power-law model

The power-law model [45] takes its inspiration from the Zipf law [61], and incorporates rank, out-degree, and eigen exponents. We used Inet [8] topology generator to obtain the power-law based Internet topologies. Briefly, Inet generates Autonomous System (AS) level topologies. These networks have similar if not the exact characteristics of the Internet from November 1997 to June 2000. The system takes in as input two parameters to generate topologies, namely: (1) the total number of nodes, and (2) the fraction (k) of degree-1 nodes. Briefly, Inet starts form the total number of desired nodes and computes the number of months t it would take to grow the Internet from its size in November 1997 (which was 3037 nodes) to the desired number of nodes. Using t it calculates the growth frequency and the out-degree of the nodes in the network. This information is used to iteratively connect nodes till the required out-degree of nodes is reached. The 20 power-law topologies were obtained using $k = \{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$.

5.2.4. Hierarchical transit–stub model

The Internet model at the AS level can also be captured by using a hierarchical model. Authors in [59] derived a graph generation method using a hierarchical model in order to provide a more adequate router model of the Internet than the Waxman model. In their paper, each AS domain in the Internet was classified as either a *transit* domain or a *stub* domain, hence the name transit–stub model. In a stub domain, traffic between any two nodes u and v goes through that domain if and only if either u or v is in that domain. Contrarily, this restriction is relaxed in a transit domain. The GT-ITM topology generator [59] models a three-level hierarchy corresponding to transit domains, stub domain, and LANs attached to stub domains [21].

Using the GT-ITM topology generator, we generated 12 random transit–stub graphs with a total of 3728 nodes each, and then placed the primary site inside a randomly selected stub domain. In order to make the topologies as realistic as possible, we introduced routing delays to mimic routers' decision delays inside the core network. We set this delay to be equal to 20 ms/hop. In order to have a realistic upper bound on the self-injected delays, the maximum hop count between any pair of sites was limited to 14 hops.

5.3. Access patterns

To evaluate the replica placement methods under realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 website [3] and NASA Kennedy Space Center website [47]. These two access logs compliment each other in many ways. The Soccer World Cup access log has over 1.35 billion requests, making it extremely useful to benchmark a given approach over a prolonged high access rate. The only drawback with these logs is that the users' IP addresses (that can potentially give us their approximate geographical locations) are replaced with an identifier. Although, we can obtain the information as to who were the top, say 500 users of the website, yet we cannot determine where the clients were from. To negate this drawback, we used the access logs collected at the NASA Kennedy Space Center website. These logs do not hide the IP addresses and thus the spatially skewed workload is preserved. Another benefit of the Space Center's log is that the access requests are very concentrated, i.e., a majority of the access request are sent from few clients (or cluster of clients)—capturing the temporal diversity of the users. This concentration is useful to benchmark the techniques over a spatially and temporally skewed workload.

An important point to note is that these logs are access (or read) logs; thus, they do not relay any information regarding the write requests. However, there is a tedious way around this. Each entry of the access logs has among other parameters, the information about the size of the object that is being accessed. The logs are processed to observe the variance in the object size. For each entry that returns the change in the object size, a mock write request is generated for that user for the object that is currently being accessed. This variance in the object size generates enough miscellanies to benchmark object updates.

5.3.1. Soccer World Cup 1998 access log

We used 88 days of the Soccer World Cup 1998 access logs, i.e., the (24 h) logs from April 30, 1998 to July 26, 1998. To process the logs, we wrote a script that returned: only those objects which were present in all the logs (from this we choose 25,000 data objects on random—the maximum workload for our experimental evaluations), the total number of requests from a particular client for an object, the average and the variance of the object size. From this log we chose the top 3728 clients (maximum experimental setup). (We will describe in Section 5.4 how we came up with the number 3728. For the time being assume that this is a correct measure.) A random mapping was then performed of the clients to the nodes of the topologies. Note that this mapping is not 1–1, rather 1–M. This gave us enough skewed workload to mimic real world scenarios. It is also worthwhile to mention that the total amount of requests entertained for each problem instance using the Soccer World Cup access logs was in the range of 3–4 million. The primary replicas' original site was mimicked by choosing random locations. The capacities of the sites $C\%$ were generated randomly with range from *Total Primary Object Sizes*/2 to $1.5 \times$ *Total Primary Object Sizes*. The variance in the object size collected from the access were used (as described in Section 5.3) to mimic the object updates. The updates were randomly pushed onto different sites, and the total system update load was measured in terms of the percentage update requests $U\%$ compared that to the initial network with no updates. For simplicity we will refer to Soccer World Cup access logs as W-log.

5.3.2. NASA Kennedy Space Center access log

We used 31 days of the NASA Kennedy Space Center access logs, i.e., the (24 h) logs from July 1, 1995 to July 31, 1995. The log has close to 1.9 million hits and 81,982 unique visitors. To process the logs, we used the same script that was used to retrieve information from the W-log, followed by the technique to inject the write accesses. The additional information regarding the IP addresses of the clients was used in conjunction with the technique proposed in [35] to map users onto the nodes of the topologies. The method described in [35] clusters the clients that are topologically close together, based on the information from the BGP routing table snapshots. (One can publicly obtain the BGP routing table information from the Looking Glass Sites Project [39] under the North American Network Operations' Group (NANOG).) For each client IP address in the access log, we find its best matching prefix in the union of all the available routing tables. All the clients whose IP addresses have the same best prefix match belong to the same cluster. A quick analysis (see Fig. 2) of this procedure shows that, the top 10, 100, 1000, and 3000 clusters accounted for about 28.98%, 54.34%, 87.03%, and 97.59% requests, respectively. From this clustering, we chose the top 3728 clusters and mapped them randomly to the 3728 nodes of the topologies. (We will describe in Section 5.4 how we came up with the number 3728. For the time being assume that this is a correct measure.) Notice that assigning a cluster, say C^i, to a node S^i in the network topology means that all the clients in C^i generate accesses from the node S^i.

Once again the primary replicas' original site was mimicked by choosing random locations. The capacities of the sites $C\%$ were generated randomly with range from *Total Primary Object Sizes*/2 to $1.5 \times$ *Total Primary Object Sizes*. The variance in the object size collected from the access were used (as described in Section 5.3) to mimic the object updates. The updates were randomly pushed onto different sites, and the total system update load was measured in terms of the percentage update requests $U\%$ compared that to the initial network with

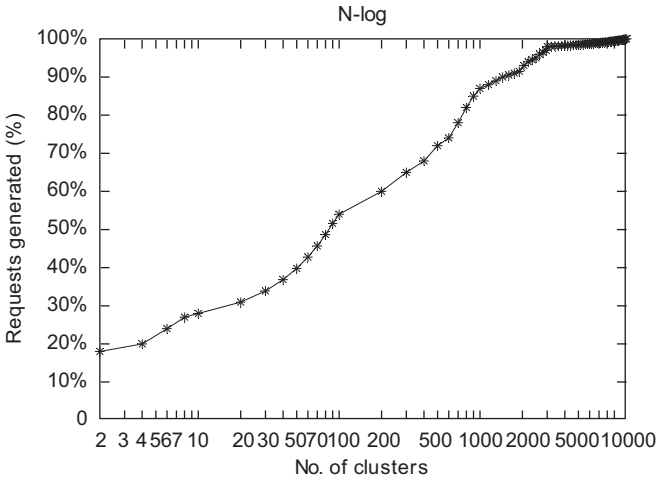

Fig. 2. Number of requests generated by the Web clusters defined by IP address prefixes.

no updates. For simplicity we will refer to NASA Kennedy Space Center access logs as N-log.

5.4. Further clarifications on the experimental setup

Since the access logs were of the year 1998 and before, we first used Inet to estimate the number of nodes in the network. This number came up to be 3728, i.e., there were 3728 AS-level nodes in the Internet at the time when the Soccer World Cup 1998 was being played. Therefore, we set the upper bound on the number of servers in the system to be $M = 3728$. Since Inet does not work for topologies before November 1997, the N-log was forward date by 2 years so that it coincided with the W-log. (We believe that this is a reasonable solution to have fair comparisons between the two logs and the underlying topologies.) Moreover, every topology model that was used in this study had the network topologies generated for $M = 3728$. Due to space limitations, we do not show the detailed results obtained using every topology. However, we do provide the averaged performance of all the comparative algorithms over all the 80 topologies and 119 (24 h) access log.

5.5. The determination of the relocation period

As noted previously (in Section 3), the time (interval t) when to initiate the replica placement techniques requires high-level human intervention. Here, we will show that this parameter if not totally can at least partially be automated. The decision when to initiate the replica placement techniques depends on the past trends of the user access patterns. Figs. 5.6.1(a) and (b) show the average (over the entire access log) user access patterns extracted from the W-log and N-log. From Fig. 5.6.1(a) we can clearly see that the Soccer World Cup 1998 website incurred soaring and stumpy traffic at various intervals during the 24-h time period (it is to be noted that the W-log has a time stamp of GMT + 1). For example, the website records its minimum requests at 0500 h. This would be an ideal time to invoke the replica placement technique(s), since the traffic is at its minimum and fewer users will be affected by the relocation of data objects in the network. Another potential time period for invoking the replica placement technique(s) is at 1800 h. In our experiments we did not use this time period since the volume of traffic at 1800 h. is enormous and it immediately soars; thus, leaving little buffer time for the completion of the replica placement technique(s).

On the other hand, the analysis of N-log (it is to be noted that the N-log has a time stamp of GMT-4) reveals two periods where the traffic drops to minimum, i.e., at 0400 h and at 2000 h. This is denoted by the two vertical lines in Fig. 5.6.1(b). Therefore, for the N-log a replica placement algorithm could be initiated twice daily: (1) at 0400 h and (2) at 2000 h. The time interval t for 0400 h would be $t = (2000–0400) = 6$ h and for 2000 h $t = (0400–2200) = 18$ h. For the W-log a replica placement algorithm could be initiated once daily at 0500 h. The time interval t for 0500 h would be $t = (0500–0500) = 24$ h.

5.6. Comparative analysis

We record the performance of the heuristics using the two access logs and 80 topologies. The plots shown in this paper are classified using the two access logs. For instance, for W-log, each point represents the average performance of an algorithm over 80 topologies and 88 days of W-log. We detail our experimental findings as follows.

5.6.1. Impact of change in the number of sites and objects

We study the behavior of the placement techniques when the number of sites increases (Figs. 4(a)–(d)), by setting the number of objects to 25,000; while in Figs. 5(a)–(d), we study the behavior when the number of objects increase, by setting the number of sites to 3718. For the first experiment we fixed $C = 15\%$ and $R/W = 0.25$. (The read/write ratio R/W reflects the relative number of reads and writes (or updates) generated for an object. For instance, $R/W = 0.25$ means that there are 25% reads and 75% writes in the system.) We intentionally chose a high workload so as to see if the techniques studied successfully handled the extreme cases.

We first study the performance of the algorithms using the W-log (Fig. 4(a)). The first observation is that WA-Star, Aε-Star, and Greedy outperform other techniques by considerable amounts. Second, GMM, SA1, SA2, SA3, and DRPA-Star fail to converge to a solution with certain problem instance. This failure to completion is directly linked to the higher ratio of writes and smaller system capacity. Some interesting observations were also observable, such as, LMM and GMM showed high gain with the initial number of site increase in the system, as much as 27% gain was recorded in case of GMM with only a 100 site increase. LMM and GMM show high initial gain since with the increase in the number of sites, the combinations of bins increase, but with the further increase in the number of sites, effect is not so observable as all the essential objects are already replicated. DRPA-Star as expected outperformed every other technique, but failed miserably, as the

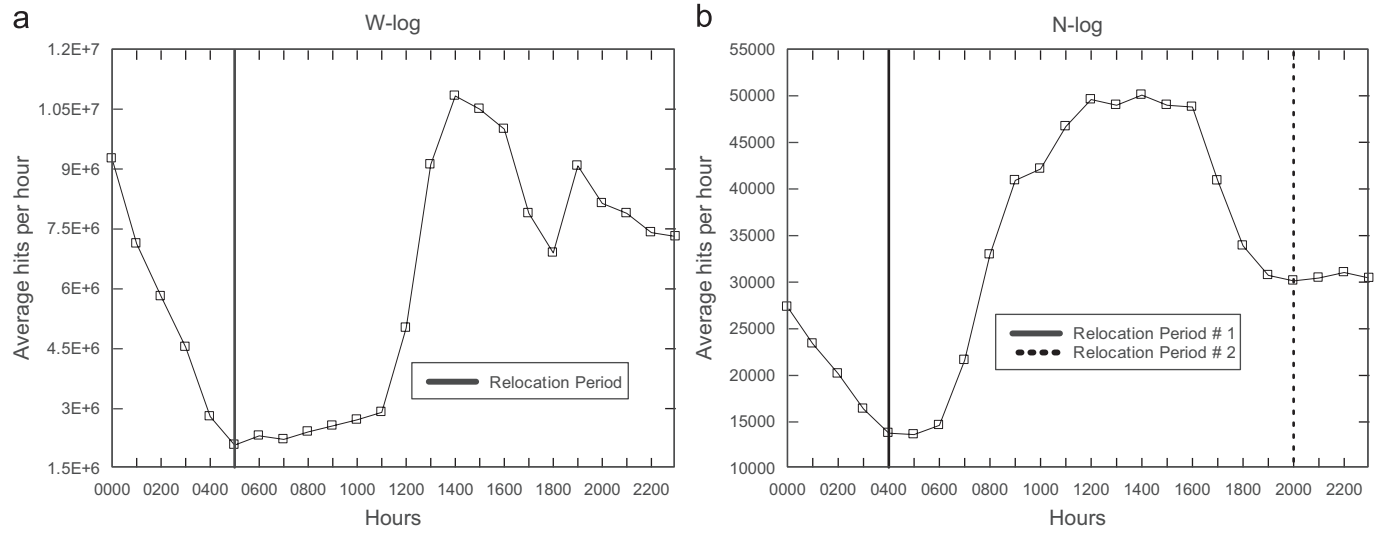

Fig. 3. User access patterns extracted from (a) W-log and (b) N-log.

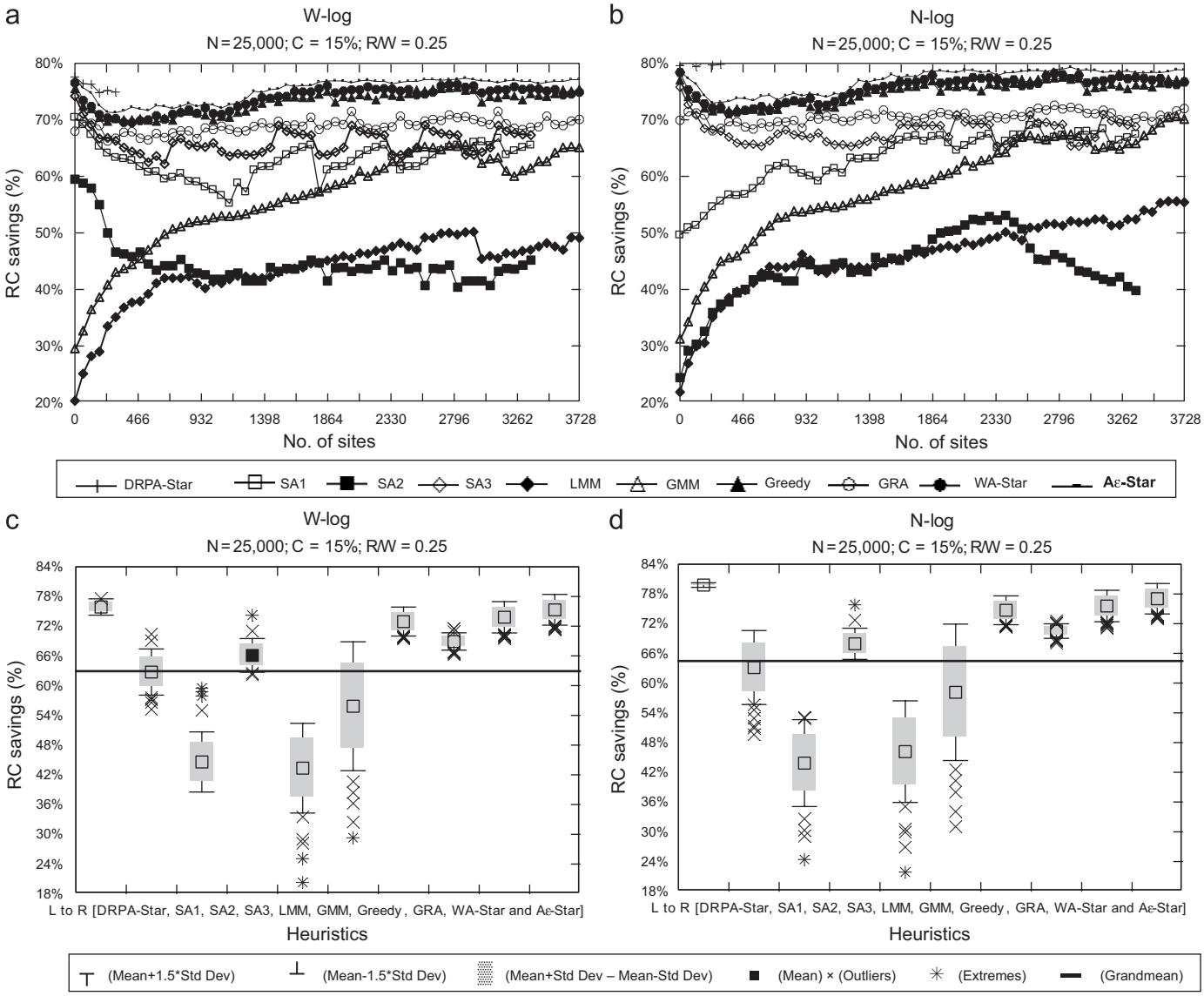

Fig. 4. (a) RC versus number of sites (W-log). (b) RC versus number of sites (N-log). (c) RC versus relative performance of heuristics (number of sites; W-log). (d) RC versus relative performance of heuristics (number of sites; N-log).

maximum workload it handled was with $M = 301$. The top performing techniques (WA-Star, Aε-Star, and Greedy) showed an almost constant performance. This is because by adding a site (server) in the network, we introduce additional traffic (local requests), together with more storage capacity available for replication. All three equally cater for the two diverse effects. GRA also showed a similar trend but maintained lower RC savings. This was in line with the claims presented in [41]. The observation made by using the N-log (Fig. 4(b)) were similar in features to that of the results obtained from the analysis of the W-log, except for the performance of SA2. The increase in the number of sites gradually decreased the RC savings of the topologies when W-log was employed; however, when N-log was used SA2 gradually increased the RC savings with the increase in the number of sites in the system. We can attribute this phenomenon to the fact that SA2 relies on the pruning of the search tree without any look-ahead technique; thus, when pruning was performed by SA2 with W-log as the workload, some useful nodes may have been pruned, resulting in the loss of RC savings. The relative performance of the algorithms pertaining to the W-log and N-log can be seen from Figs. 4(c) and (d), respectively. The plots show the mean performance of the algorithms, with bars at the maximum and minimum limits with values of mean $+1.5$ times the standard deviation and mean -1.5 times the standard deviation, respectively. The shaded block represents the maximum and minimum limits with values of mean + standard deviation and mean − standard deviation, respectively. The solid line across the plots is the grand mean, the solid block (■) represents the mean, the cross ($\times$) represents the outliers, and the asterisk ($*$) denotes the extremes. We limit the outliers and extremes to 2 and 3 standard deviations, respectively. The plots are self-explanatory and show exactly which algorithms provide high (consistent) performance. The performance of the techniques based on the RC versus number of sites criteria are ranked as follows: (1) DRPA-Star; (2) Aε-Star; (3) WA-Star; (4) Greedy; (5) GRA; (6) SA3; (7) SA1; (8) GMM; (9) SA2; (10) LMM.

To observe the effect of increase in the number of objects in the system, we chose a softer workload with $C = 45\%$ and $R/W = 0.75$. The intention was to observe the trends for all the algorithms as much as possible as some techniques failed to yield results as observable from Figs. 4(a) and (b). Moreover, we want to observe the algorithms under various (system) environments. The increase in the number of objects has diverse effects on the system as new read/write patterns (users are offered more choices) emerge, and also the increase in the strain on the overall capacity of the system (increase in the number of replicas). An effective algorithm should incorporate both the opposing trends.

We first observe the performance of the algorithms using the W-log (Fig. 5(a)). From the plot, we can observe that the bin packing techniques perform the worst with a loss of nearly 32% in case of LMM. The most surprising result came from GRA. It dropped its savings from 62% to 12%. This was contradictory to what was reported in [41]. But there the authors had used a uniformly distributed link cost topology, and their traffic was based on the Zipf distribution [61]. While the traffic access logs of the World Cup 1998 are more or less double-Pareto in nature [33]. In either case the exploits and limitations of the technique under discussion are obvious. The plot also shows a near identical performance by WA-Star, Aε-Star, and Greedy. The relative difference among the three techniques is less than 5%. However, Aε-Star did maintain its supremacy.

With N-log (Fig. 5(b)), the relative performance of the algorithms dropped further, this is due to the fact that the N-log is highly concentrated. An increase in the number of objects increases the traffic in the system by multi-folds, and the RC savings drop since the algorithms cannot further identify placements for the newly introduced objects. However, this drop in RC savings is not more than 5–10% compared with that of the results obtained from W-log. To better understand this phenomenon, readers are encouraged to examine the relative trends observable from Figs. 5(c) and (d). The performance of the techniques based on the RC versus number of objects criteria are ranked as follows: (1) DRPA-Star; (2) Greedy; (3) Aε-Star; (4) WA-Star; (5) SA3; (6) SA1; (7) SA2; (8) GRA; (9) GMM; (10) LMM.

From here onwards, we will not report the performance of DRPA-Star, as it is only effective and converges to a solution when the problem size is considerably small. However, we will log the DRPA-Star algorithm termination timings (on small problem instances). Moreover, we will give a default first ranking to the DRPA-Star in the subsequent text since it always produces an optimal solution.

5.6.2. Impact of change in the system capacity

An increase in the storage capacity means that a large number of objects can be replicated. Replicating an object that is already extensively replicated, is unlikely to result in significant traffic savings as only a small portion of the servers will be affected overall. Moreover, since objects are not equally read intensive, increase in the storage capacity would have a great impact at the beginning (initial increase in capacity), but has little effect after a certain point, where the most beneficial ones are already replicated.

We first observe the performance of the algorithms using the W-log (Fig. 6(a)). LMM and GMM once again performed the worst. The gap between all other approaches was reduced to within 12% of each other. WA-Star and Aε-Star showed an immediate initial increase (the point after which further replicating objects is inefficient) in its RC savings, but afterward showed a near constant performance. GRA observable gained the most RC savings 38% followed by Greedy with 31%.

Near identical performances were recorded using the N-log (Fig. 6(b)). One interesting observation is that when the system capacity is increased from 28% to 30%, the relative performance of almost all the algorithms increase by at most 10%. This sudden increase in RC savings can be linked to the spatially distributed access of the clients in the N-log. That is, the 28% system capacity was marginally small to place the needed replicas in the vicinity of the clients so that the relative communication cost is minimized; thus, with only an increase of 2%, all the critically required replicas could now be placed and hence the sudden surge in RC savings.

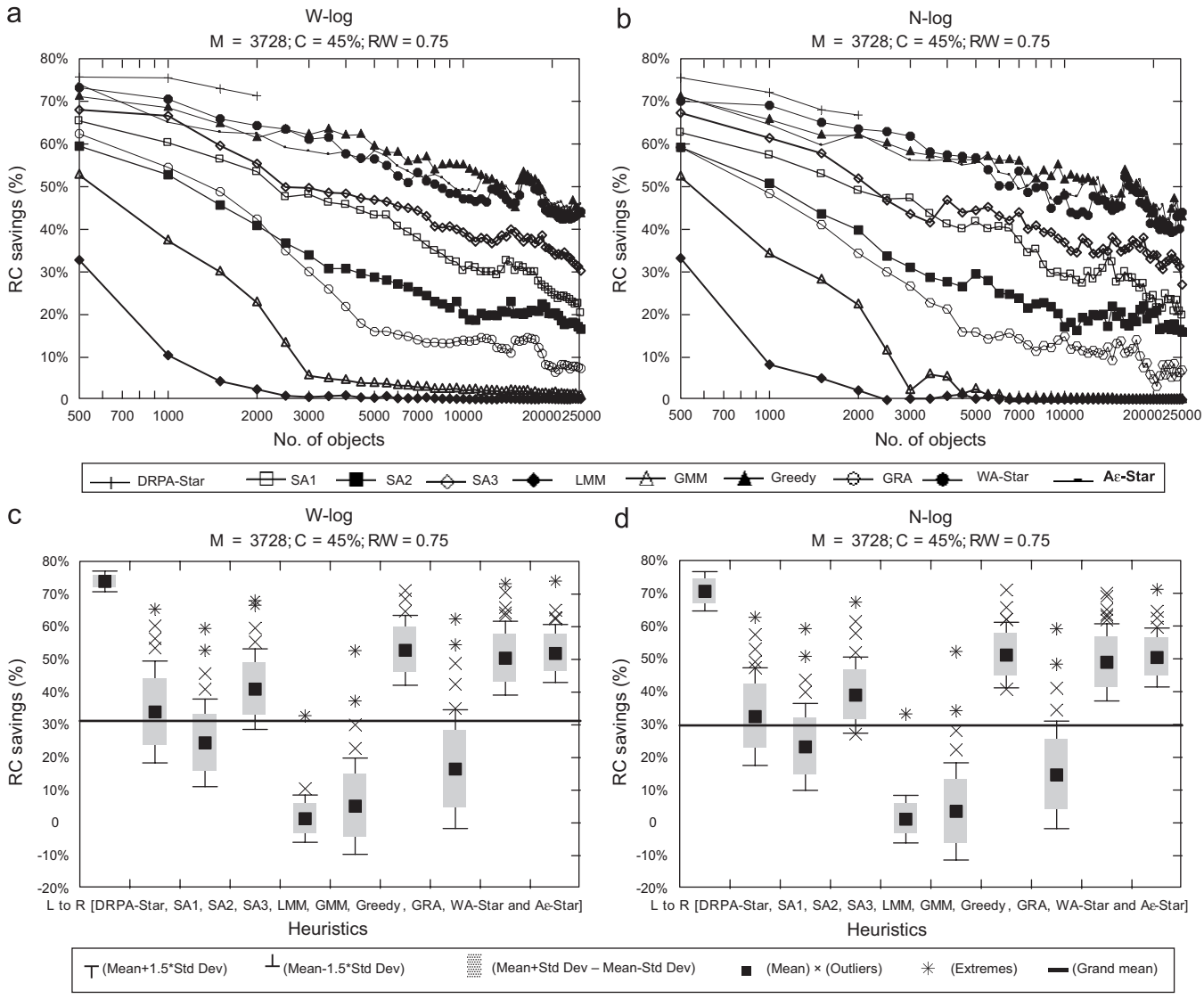

Fig. 5. (a) RC versus number of objects (W-log). (b) RC versus number of objects (N-log). (c) RC versus relative performance of heuristics (number of objects; W-log). (d) RC versus relative performance of heuristics (number of objects; N-log).

Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy that the increase in capacity from 10% to 17% resulted in four times (on average) more replicas for all the algorithms. The performance of the techniques based on the RC versus system capacity criteria (and by observing Figs. 6(c) and (d)) are ranked as follows: (1) DRPA-Star; (2) Aε-Star; (3) Greedy; (4) WA-Star; (5) GRA; (6) SA3; (7) SA1; (8) SA2; (9) GMM; (10) LMM.

5.6.3. Impact of change in the read/write frequencies

Since the read and write parameters are complementary to each other, we take the liberty to describe them together. In both the setups the number of sites and objects were kept constant. Increase in the number of reads in the system would mean that there is a need to replicate as many object as possible (closer to the users). However, the increase in the number of updates (or writes) in the system requires the replicas be placed as close as to the primary site as possible (to reduce the update broadcast). This phenomenon is also interrelated with the system capacity, as the update ratio sets an upper bound on the possible traffic reduction through replication. Thus, if we consider a system with unlimited capacity, the "replicate everywhere anything" policy is strictly inadequate. The read and update parameters indeed help in drawing a line between good and marginal algorithms.

Figs. 7(a) and (b) show the performance of the algorithms using the W-log and the N-log, respectively. A clear classification can be made between the algorithms. WA-Star, Aε-Star, and Greedy incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 90%. LMM gained the least of the RC savings of: up to 39% with the W-log and up to 36% with the N-log. However, the performance of LMM and GMM decreased exponentially with the increase in R/W ratio. A sub-exponential decrease was also observable

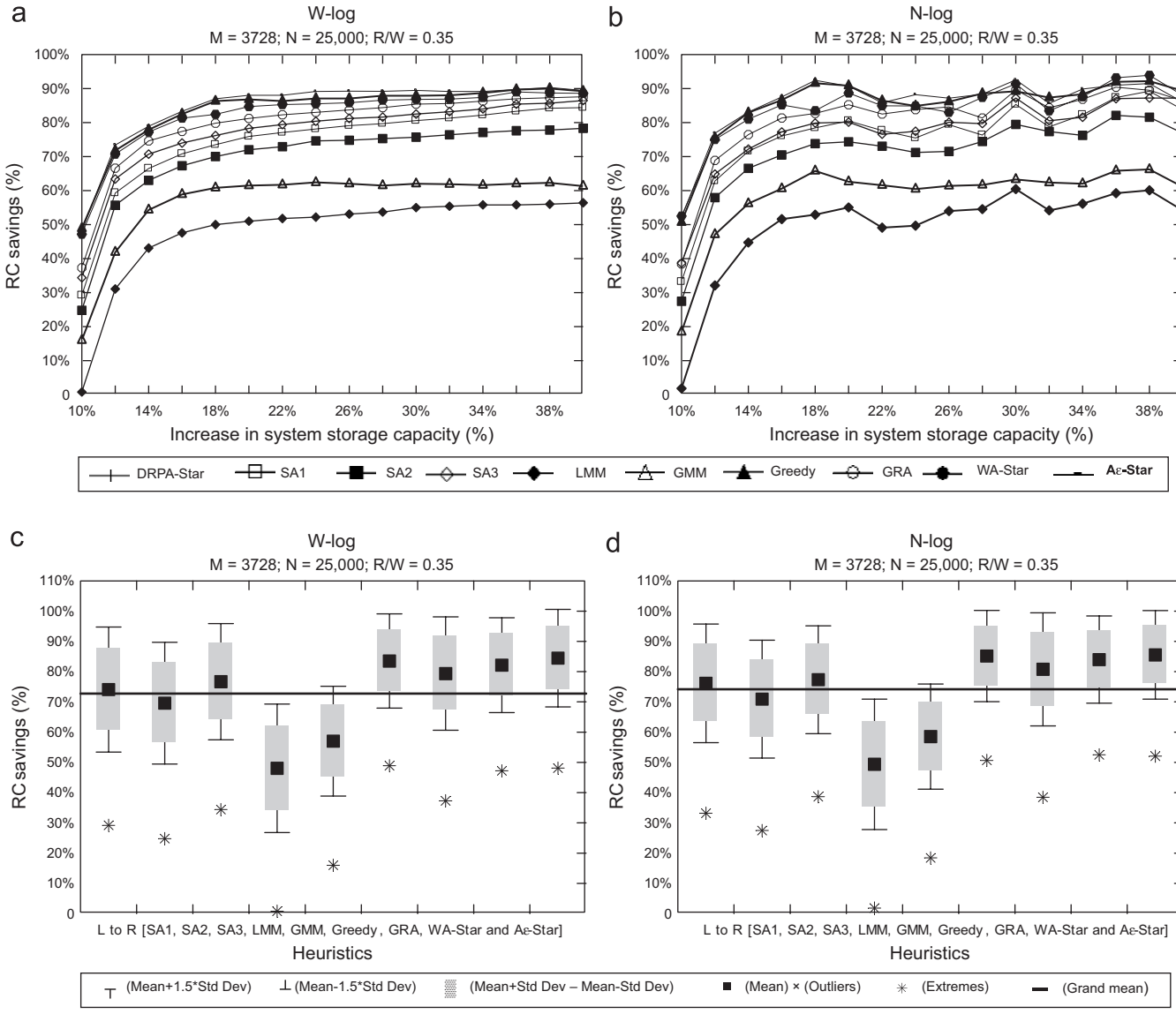

Fig. 6. (a) RC versus system capacity (W-log). (b) RC versus system capacity (N-log). (c) RC versus relative performance of heuristics (system capacity; W-log). (d) RC versus relative performance of heuristics (system capacity; N-log).

in the case of GRA. (All algorithms exhibited some sort of decrease in RC savings with R/W ratio of 0.50 and above.) WA-Star, Aε-Star, and Greedy on the other hand showed extreme robustness and retained their initial savings. To understand why there is such a gap in the performance between the algorithms, we should recall that LMM and GMM specifically exploit the capacities of the servers, while the optimization of the RC is a secondary consideration. Moreover, they maintain localized network perceptions. Increase in updates result in objects having decreased local significance (unless the vicinity is in close proximity to the primary location). On the other hand, the sub-optimal A-Star based heuristics suffer from the bound set for their search tree. Thus, by definition they tend to optimize local replication. However, WA-Star, Aε-Star, and Greedy never tend to deviate from their global view of the problem search space. To better understand this phenomenon, readers are encouraged to examine the relative trends observable from Figs. 7(c) and (d). The performance of the techniques based on the RC versus R/W ratio criteria are ranked as follows: (1) DRPA-Star; (2) Aε-Star; (3) WA-Star; (4) Greedy; (5) SA3; (6) GRA; (7) SA1; (8) GMM; (9) SA2; (10) LMM.

5.6.4. Running time

Before we proceed with the discussion, we would like to clarify our measurement of algorithm termination timings. The approach we took was to see if these algorithms can be used in dynamic scenarios. Thus, we gather and process data as if it was a dynamic system. The average breakdown of the execution time of all the algorithms combined is depicted in Fig. 8. There 68% of all the algorithm termination time was taken by the repeated calculation of the shortest paths. Data gathering and dispersion, such as reading the read frequencies

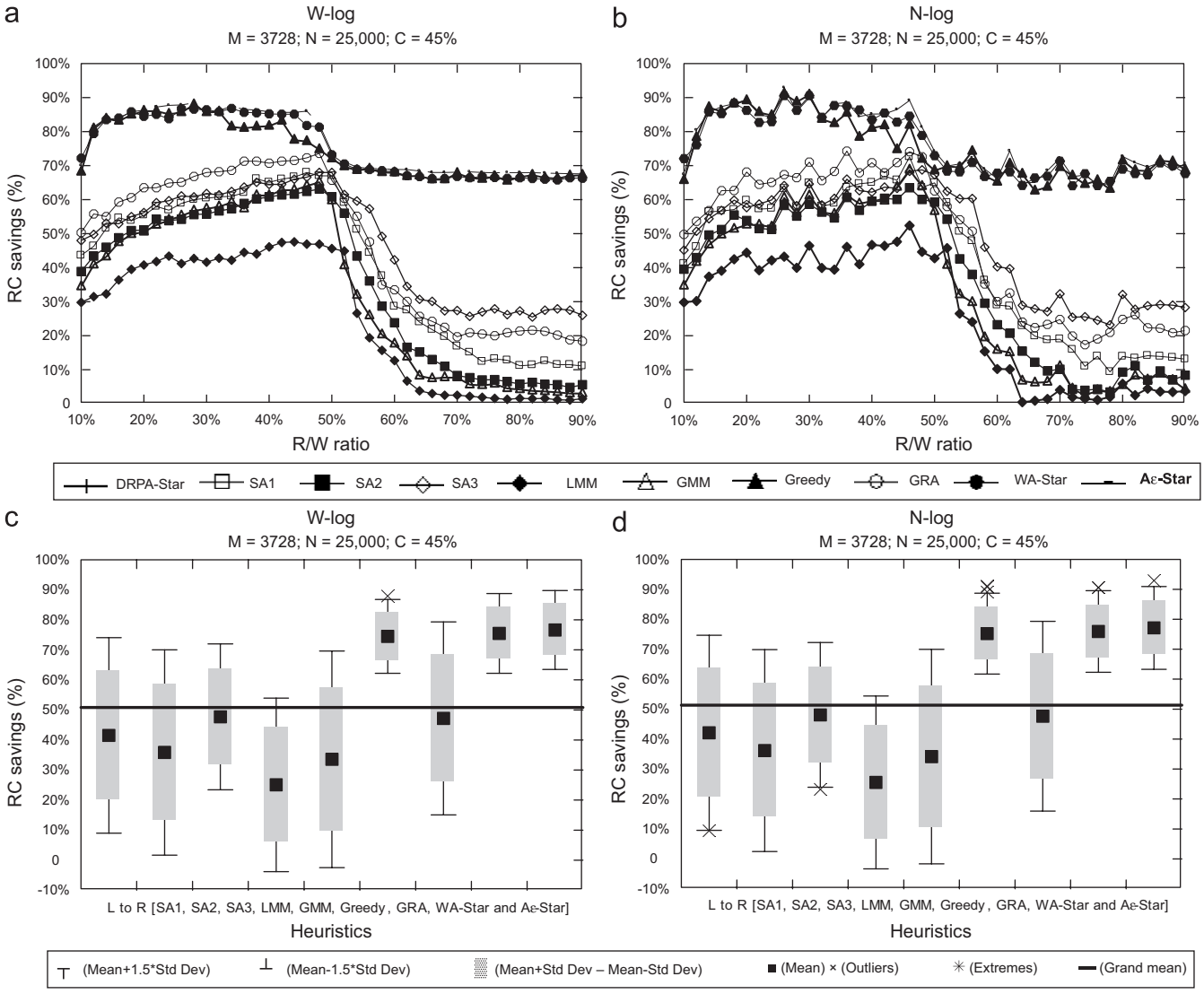

Fig. 7. (a) RC versus R/W ratio (W-log). (b) RC versus R/W ratio (N-log). (c) RC versus relative performance of heuristics (R/W ratio; W-log). (d) RC versus relative performance of heuristics (R/W ratio; N-log).

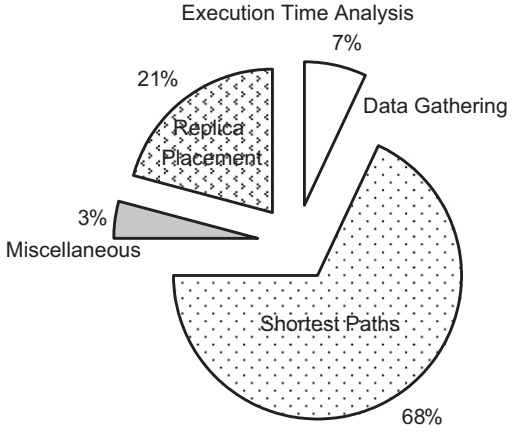

Fig. 8. Execution time components.

from the processed log, etc., took 7% of the total time. Other miscellaneous operations including input/output were recorded to carry 3% of the total execution time. From the plot it is clear that a totally static setup would take no less than 21% of the time depicted in Tables 4(a)–(c).

Various problem instances were recorded with $C = 20\%, 45\%, 75\%$ and $R/W = 0.55, 0.65, 0.85$. Each problem instance represents the average recorded time over all the 80 topologies and 119 various access logs. The entries in bold represent the fastest time recorded over the problem instance. It is observable that LMM terminated faster than all the other techniques, followed by Greedy and GMM. If a static environment was considered, LMM with the maximum problem instance would have terminated approximately in 317.94 s (21% of the algorithm termination time). An interesting result is also observable in the cases of SA1 and SA2. With soft

Table 4
Running time in seconds: (a) $C = 20\%$, $R/W = 0.55$ (small problem instances); (b) $C = 45\%$, $R/W = 0.85$ (medium problem instances); (c) $C = 75\%$, $R/W = 0.65$ (large problem instances)

Problem size	DRPA-Star	SA1	SA2	SA3	LMM	GMM	Greedy	GRA	WA-Star	Aε-Star
$M = 20, N = 50$	274.02	95.32	101.96	116.00	**67.20**	72.07	70.47	92.25	104.63	97.59
$M = 20, N = 100$	315.73	103.32	111.04	120.96	80.31	78.38	**77.25**	97.66	110.21	104.02
$M = 20, N = 150$	351.55	120.90	122.19	158.14	97.81	90.48	**79.67**	102.72	134.04	114.49
$M = 30, N = 50$	365.04	136.90	158.61	175.64	100.55	105.25	**96.11**	128.63	149.15	142.71
$M = 30, N = 100$	389.77	143.27	178.62	198.66	**105.23**	116.33	109.48	126.25	173.80	149.22
$M = 30, N = 150$	469.23	184.84	206.05	237.70	**115.17**	130.83	137.65	150.33	210.82	180.66
$M = 40, N = 50$	578.48	185.38	259.89	279.69	**117.78**	136.16	128.12	155.59	251.95	200.25
$M = 40, N = 100$	706.89	234.98	308.06	325.29	**120.81**	158.93	135.07	169.17	288.12	237.93
$M = 40, N = 150$	957.41	248.23	359.76	365.57	**122.81**	165.23	141.92	205.61	325.18	272.43

Problem size	SA1	SA2	SA3	LMM	GMM	Greedy	GRA	WA-Star	Aε-Star
$M = 300, N = 1350$	292.77	205.13	226.01	177.58	195.17	**189.66**	243.04	242.02	248.20
$M = 300, N = 1400$	310.17	203.38	247.80	198.26	**205.65**	206.61	327.70	253.55	280.64
$M = 300, N = 1450$	317.00	236.94	258.35	**207.38**	234.46	238.80	381.24	270.32	311.64
$M = 300, N = 1500$	328.51	261.78	272.72	**248.21**	260.18	259.81	410.46	288.71	334.56
$M = 300, N = 1550$	358.53	280.17	289.49	**269.18**	276.96	276.22	469.88	309.70	370.30
$M = 300, N = 2000$	391.03	297.39	310.84	276.38	306.66	**269.19**	477.18	333.96	388.16
$M = 400, N = 1350$	405.11	310.38	359.36	**306.98**	347.67	323.82	494.62	358.66	354.13
$M = 400, N = 1400$	429.54	327.47	404.21	**325.15**	349.07	349.31	536.83	386.89	369.04
$M = 400, N = 1450$	460.38	361.57	440.94	**360.97**	370.23	368.19	543.05	421.74	397.92
$M = 400, N = 1500$	487.87	373.85	469.19	376.31	**375.98**	378.72	560.49	443.86	413.91
$M = 400, N = 1550$	499.61	359.76	496.88	**381.46**	389.77	389.93	606.75	442.29	415.83
$M = 400, N = 2000$	537.82	390.36	510.35	412.82	**392.25**	418.78	660.13	479.12	448.45
$M = 500, N = 1350$	560.63	389.90	527.54	429.82	433.42	**402.84**	661.87	492.33	460.61
$M = 500, N = 1400$	610.79	469.65	610.35	456.25	479.07	**454.98**	690.45	564.89	513.31
$M = 500, N = 1450$	663.70	584.08	664.18	**472.05**	486.13	503.05	705.96	637.70	582.85
$M = 500, N = 1500$	707.04	643.08	741.38	**498.35**	510.96	532.92	736.81	698.34	627.87
$M = 500, N = 1550$	806.50	700.24	809.32	**503.97**	526.75	584.71	754.96	771.25	646.49
$M = 500, N = 2000$	847.04	725.58	903.17	**518.85**	539.35	636.19	778.28	826.30	736.46

Problem size	SA1	SA2	SA3	LMM	GMM	Greedy	GRA	WA-Star	Aε-Star
$M = 2500, N = 15,000$	930.52	612.96	744.10	**598.33**	618.51	622.05	983.82	765.60	842.32
$M = 2500, N = 20,000$	957.44	715.01	779.93	**629.17**	706.22	724.42	1148.09	814.95	940.19
$M = 2500, N = 25,000$	1178.41	898.60	937.30	836.55	925.40	**808.91**	1438.01	1006.75	1167.34
$M = 3000, N = 15,000$	1290.70	986.60	1215.96	**975.73**	1050.77	1049.76	1613.73	1162.22	1109.52
$M = 3000, N = 20,000$	1467.07	1128.04	1412.25	1136.03	**1131.15**	1139.63	1683.45	1337.60	1248.56
$M = 3000, N = 25,000$	1685.60	1173.15	1584.43	1290.00	1279.38	**1209.19**	1986.36	1477.87	1382.90
$M = 3718, N = 15,000$	1837.25	1413.91	1836.77	1372.30	1443.06	**1370.55**	2078.63	1702.54	1544.59
$M = 3718, N = 20,000$	2120.34	1929.82	2224.72	**1495.99**	1534.47	1601.01	2211.73	2098.41	1886.13
$M = 3718, N = 25,000$	2423.99	2104.62	2432.76	**1514.02**	1587.18	1760.80	2271.37	2319.41	1945.11

problem instances, SA1 terminates faster than SA2, but the trend is reversed, when the algorithms tackle hard problem instances. This is because with smaller problem instance SA2 has an extra overhead of discarding nodes from the OPEN list.

5.6.5. Summary of performances

In summary, based on the solution quality alone, the algorithms can be classified into four categories: (1) the high performance algorithms that include DRPA-Star, Aε-Star, WA-Star, and Greedy; (2) the medium-high performance algorithms of GRA and SA3; (3) the medium performance algorithms of SA1 and SA2; (4) the low performance algorithms of GMM and LMM. While considering the termination timings, LMM, GMM, and Greedy did extremely well, followed by Aε-Star, SA2, WA-Star, SA1, and SA3. DRPA-Star as expected finished at the bottom of the list courtesy to its sub-exponential running time.

5.7. Supplementary results

Here, we present some supplementary results that strengthen our comparative analysis reported in Section 5.5. The relative performance of the heuristics with variance in R/W ratio and system storage capacity are shown in Figs. 9(a) and (b), respectively. The main idea behind these two plots was to show the relative performance of all the algorithms over every possible combination over all the 80 topologies and 119 access logs. In both the cases, we fix $M = 3728$ and $N = 25,000$. The variance for the R/W ratio was measured between $R/W =$ [0.1–0.90], and the variance for the storage capacity was

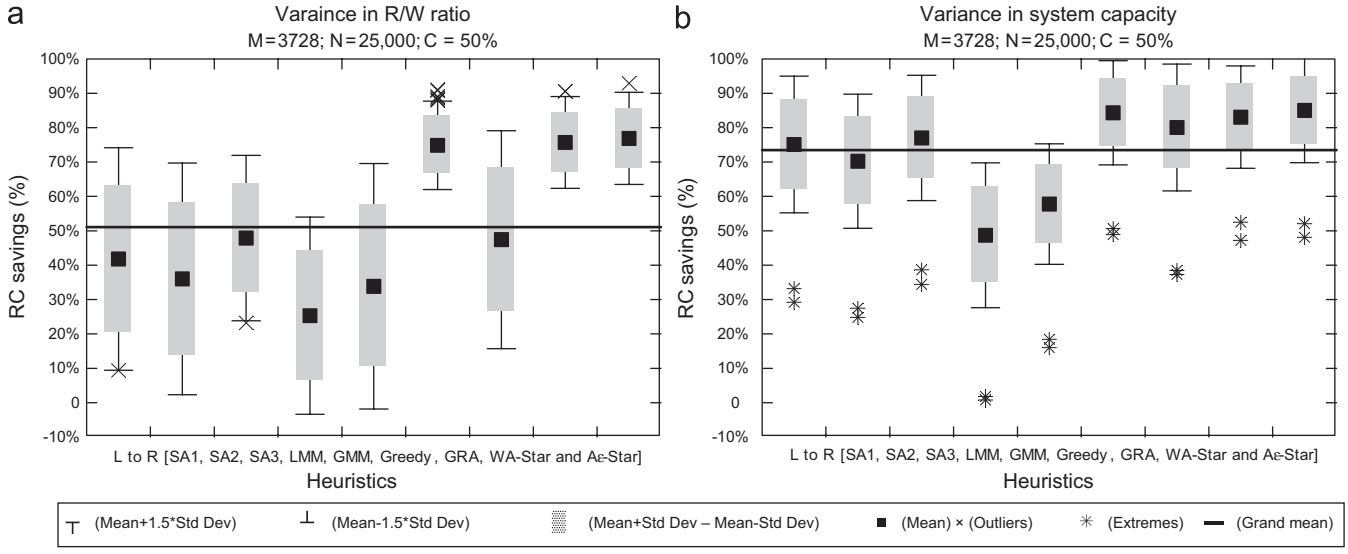

Fig. 9. (a) RC versus variance in R/W ratio. (b) RC versus variance in system capacity.

Table 5
Problem instances for recording search tree node expansion savings

Serial no.	Problem instance	Serial no.	Problem instance
1	$M = 20, N = 50$	6	$M = 30, N = 150$
2	$M = 20, N = 100$	7	$M = 40, N = 50$
3	$M = 20, N = 150$	8	$M = 40, N = 100$
4	$M = 30, N = 50$	9	$M = 40, N = 150$
5	$M = 30, N = 100$	10	$M = 50, N = 200$

measured between $C = [20–80\%]$. The plots show the mean performance of the algorithms, with bars at the maximum and minimum limits with values of mean $+1.5$ times the standard deviation and mean -1.5 times the standard deviation, respectively. The shaded block represents the maximum and minimum limits with values of mean + standard deviation and mean − standard deviation, respectively. The solid line across the plots is the grand mean, the solid block (■) represents the mean, the cross ($\times$) represents the outliers, and the asterisk ($*$) denotes the extremes. The outliers and extremes are limited to 2 and 3 standard deviations, respectively. We are mostly interested in measuring the mean interval.

With R/W variance (Fig. 9(a)), Aε-Star edges over WA-Star with a savings of 78%. Although Greedy recoded the highest RC savings (94%) its mean interval was around 76%. Among the suboptimal A-Star heuristics SA2 showed a very stable performance but SA3 recorded a higher mean interval. LMM and GMM observably performed the worst. Fig. 9(b) depicts the adaptability of the algorithms under the variance of the system storage capacity. It can be seen that all the algorithms did well in this domain. Unexpectedly, LMM and GMM which basically only exploit the storage capacity could not show a performance comparable to their counterparts. We attribute this fact to the arguments presented earlier—the bin packing algorithms only focus on local network optimization and do not have the full picture of the problem domain. Once again, Aε-Star edges over WA-Star, which is closely followed by Greedy.

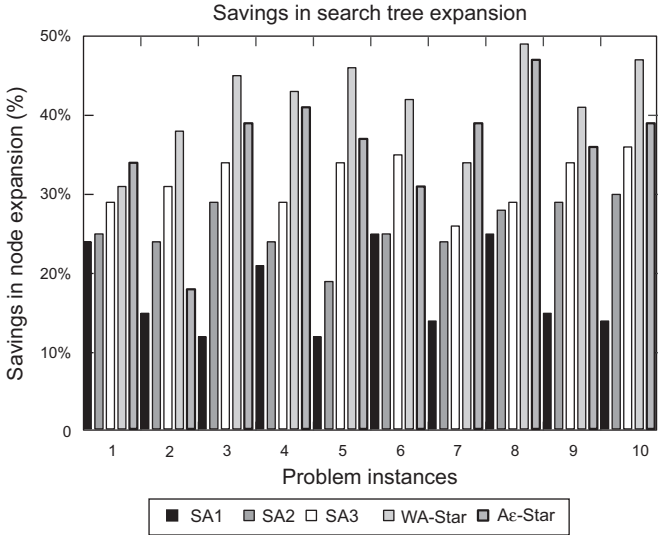

Fig. 10. Search tree node expansion savings of A-Star based heuristics.

Lastly, we compare the pruning strength of the A-Star based heuristics. Ten problem instances (Table 5) were put to test. Fig. 10 shows the savings in the node expansion compared to that of DRPA-Star. Clearly WA-Star prunes more nodes than any of the other heuristics, followed by Aε-Star, SA3, SA2, and SA1.

5.8. Recap of results

Table 6 reports the solution quality in terms of RC percentage for 20 randomly chosen problem instances, each being a combination of various numbers of sites and objects, with varying storage capacity and R/W ratio. For each row, the best result

Table 6
Average RC (%) savings under some problem instances

Problem size	DRPA-Star	SA1	SA2	SA3	LMM	GMM	Greedy	GRA	WA-Star	Aε-Star
$M = 20, N = 150$ [$C = 20\%$, $R/W = 0.75$, W-log]	**79.72**	75.70	68.20	77.66	50.90	63.38	74.62	73.01	76.96	78.54
$M = 50, N = 200$ [$C = 20\%$, $R/W = 0.80$, N-log]	**80.59**	72.69	73.15	74.17	43.59	59.48	67.44	72.62	77.24	76.75
$M = 50, N = 300$ [$C = 25\%$, $R/W = 0.95$, N-log]	**77.17**	73.47	71.42	72.71	49.88	66.33	75.34	74.36	73.45	71.37
$M = 60, N = 300$ [$C = 35\%$, $R/W = 0.95$, W-log]	**74.07**	67.32	66.08	71.46	45.38	62.70	69.79	70.39	69.04	72.45
$M = 100, N = 400$ [$C = 25\%$, $R/W = 0.75$, W-log]	X	72.22	68.00	**73.91**	46.56	67.73	67.44	70.17	73.68	72.37
$M = 100, N = 500$ [$C = 30\%$, $R/W = 0.65$, W-log]	X	66.68	64.15	69.44	47.70	64.30	69.07	64.95	**70.83**	70.00
$M = 200, N = 800$ [$C = 25\%$, $R/W = 0.85$, W-log]	X	**73.49**	70.93	72.12	50.59	67.85	72.08	69.93	72.80	71.71
$M = 100, N = 1000$ [$C = 20\%$, $R/W = 0.95$, W-log]	X	76.62	67.98	**79.39**	54.61	67.60	78.91	71.75	75.68	71.73
$M = 300, N = 1000$ [$C = 25\%$, $R/W = 0.75$, N-log]	X	69.32	66.06	61.27	58.18	65.53	74.35	65.75	74.23	**75.57**
$M = 400, N = 1500$ [$C = 35\%$, $R/W = 0.60$, N-log]	X	68.10	68.08	69.15	53.80	56.86	71.55	64.59	72.19	**72.97**
$M = 200, N = 2000$ [$C = 20\%$, $R/W = 0.80$, W-log]	X	72.97	68.53	74.01	53.94	63.50	72.67	71.96	75.71	**77.20**
$M = 500, N = 2000$ [$C = 60\%$, $R/W = 0.40$, N-log]	X	69.93	67.48	71.75	47.35	63.73	72.10	77.93	79.50	**81.95**
$M = 500, N = 3000$ [$C = 25\%$, $R/W = 0.95$, W-log]	X	73.14	69.34	**73.74**	50.30	68.10	72.32	71.09	73.35	71.36
$M = 1000, N = 5000$ [$C = 35\%$, $R/W = 0.95$, N-log]	X	66.20	67.74	67.20	44.94	62.41	70.10	70.43	71.86	**71.87**
$M = 1500, N = 10,000$ [$C = 25\%$, $R/W = 0.75$, N-log]	X	75.10	66.59	**78.34**	44.49	68.00	72.06	69.79	72.93	74.41
$M = 2000, N = 15,000$ [$C = 30\%$, $R/W = 0.65$, W-log]	X	76.80	73.86	**81.22**	51.04	68.37	72.84	68.03	73.06	75.64
$M = 2500, N = 15,000$ [$C = 25\%$, $R/W = 0.85$, N-log]	X	68.13	66.79	**70.60**	46.61	62.37	69.61	67.44	69.55	67.45
$M = 3000, N = 20,000$ [$C = 30\%$, $R/W = 0.65$, W-log]	X	74.67	72.19	**74.75**	51.90	66.63	65.54	70.68	74.49	73.99
$M = 3500, N = 25,000$ [$C = 35\%$, $R/W = 0.50$, W-log]	X	72.33	69.22	73.22	56.22	62.29	72.70	66.70	72.93	**74.20**
$M = 3718, N = 25,000$ [$C = 65\%$, $R/W = 0.40$, N-log]	X	67.07	65.78	68.67	55.56	60.19	70.38	64.29	**71.95**	71.26

Table 7
Algorithm ranking based on solution quality

Algorithm	RC savings				Overall score	Rankings
	Sites	Objects	Capacity	R/W		
DRPA-Star	1	1	1	1	4	1
SA1	7	6	7	7	27	7
SA2	9	7	8	9	33	8
SA3	6	5	6	5	22	5
LMM	10	10	10	10	40	10
GMM	8	9	9	8	34	9
Greedy	4	2	3	4	13	3
GRA	5	8	5	6	24	6
WA-Star	3	4	4	3	14	4
Aε-Star	2	3	2	2	9	2

Table 8
Overview of results with suggested utilization

Algorithm	Running time	Memory utilization	Solution quality	Suggested utilization
DRPA-Star	Very high	Very high	Optimal	Static with optimal quality (not practical at all)
SA1	Medium–high	Medium	Medium	Static with medium quality
SA2	Medium	Medium	Medium–high	Static/dynamic with medium–high quality
SA3	High	Medium	Medium–high	Static with medium–high quality
LMM	Very low	Very low	Low	Fast/dynamic with low quality
GMM	Low–medium	Low	Low–medium	Fast/dynamic with low–medium quality
Greedy	Low	Low	High	Fast/dynamic with high quality
GRA	Medium	Medium	Medium–high	Static with medium–high quality
WA-Star	Medium	Low–medium	High	Static/dynamic with high quality
Aε-Star	Medium	Low–medium	High	Medium fast/dynamic with high quality

is indicated in bold. Entries marked with "X" represent that the algorithm could not be terminated in a reasonable time. A-Star (DRPA-Star) and A-Star based heuristics (Aε-Star, WA-Star, SA1, SA2, SA3) steal the show in the context of solution quality, but Greedy and GRA do indeed give a good competition, with a savings within a range of 5–10% of Aε-Star.

As we mentioned in the introductory passage, selecting the best heuristic to be used in a given environment, is a difficult

task, unless thorough investigation of the heuristics under a unifying problem domain is performed. For this purpose we selected 10 heuristics from literature and extensively compared them under the variance of various system parameters. This study gives us the confidence to select a heuristic given a well-defined environment. Based on our findings, we are now in a position, where we can declare a "winner" among all the studied techniques. We poll our vote in favor of the Greedy heuristics which was originally proposed in [50]. Although, it finished second on termination time and third (counting DRPA-Star which is only effective with smaller problem instances) in terms of the solution quality, yet its solution quality was always within 2–5% of Aε-Star. Moreover, test results showed that Greedy was considerably faster than the other high performing algorithms. Table 7 shows the numerical ranking of the algorithms based on the solution quality. The overview of results and our recommendations on the possible usage of the heuristics studied in this paper are summarized in Table 8.

6. Concluding remarks

Replicating data across a network can be instrumental in reducing the user perceived access time. In this paper we compared and analyzed 10 static heuristics-based fine-grained (object based) data replication techniques. Selecting the best heuristic to be used in a given environment, however, remains a difficult task, since comparisons are often clouded by different underlying assumptions in the original study of the heuristic. The main purpose of this paper was to study and compare heuristics on a unifying platform with changing system parameters so as to fully understand the capabilities and limitations of the methods. The studied heuristics were thoroughly tested using an experimental test-bed that closely mimicked the Internet in its infrastructure and user access patterns. GT-ITM and Inet Internet topology generators were used to obtain 80 well-defined network topologies based on flat, link distance, power-law and hierarchical transit–stub models. The user access patterns were derived from real access logs collected at the Soccer World Cup 1998 web-server and NASA Kennedy Space Center web-server. The selection of two different access logs was necessary to compliment the pros and cons of each access log. The Soccer World Cup 1998 access log had a high volume of traffic but did not cater for the geographically distributed access load. On the other hand, the NASA Kennedy Space Center access log had diverse spatial and temporal access requests. Using this setup, the heuristics were evaluated by analyzing the system utilization in terms of reducing the communication cost incurred due to object transfer(s) under the variance of server capacity, object size, read access, write access, number of objects and sites. Based on our observations, we made detailed suggestive uses of each and every heuristic and identified the circumstances in which they are deemed useful.

Our main observations are as follows:

1. Replica placement algorithms should incorporate the knowledge of network topology, clients' access requirements, and possibly the psychological aspects that affect the access frequencies.
2. Care should be taken when to invoke a replica placement algorithm. Studying the past user access trends can potentially help the designers to effectively determine the (exact) time when to invoke the algorithm.
3. The relative performance of the replica placement algorithms is not the same across the access logs (Soccer World Cup 1998 and NASA Kennedy Space Center), network topologies (flat, link distance, power-law, and transit–stub), system parameters (capacity, read/write ratio, number of sites and objects). It would be extremely useful to identify the environment before the choice of deploying a particular algorithm is undertaken.
4. As demonstrated in this study, there is no single replica placement algorithm that can cater for all the possible scenarios. This study can be used as a benchmark for selecting algorithms (proposed in this paper or otherwise) to effectively tackle the underlying system environment.

Our suggested line of action for the content distributors or network managers who would like to incorporate this study in their network management portfolio is as follows:

1. Obtain the approximate (if not the exact) network topology. This may be a daunting task, but nonetheless an approximate measure of the network topology would be extremely useful in obtaining a closed form cost model.
2. Gather all the system parameters such as, system storage capacity, number of objects and sites. This is relatively an easy task, since the content distributors or network managers would have first-hand information about the number of objects that they want to replicate and the number of potential sites onto which they expect replication. Once the number of sites in the system is known, the system storage capacity would simply be an aggregation operation.
3. Observe the access patterns of the clients and extract the spatially and temporally distributed workload so that read/write ratios in correspondence to the system parameters can be obtained. This task can be performed by processing the access logs to reflect their spatial and temporal properties and project the write access using methods (IP clustering, relative object size variance) articulated in this paper.
4. Using system parameters and access patterns determine the relocation time for the replica placement algorithm based on the simple aggregate access graph method.

Once the above stated four steps are undertaken, the selection of a preferred heuristic (or a set of heuristics) should exhibit balance between: (1) execution time, (2) memory utilization, (3) solution quality, and (4) adaptation for altering system parameters.

If frequent replica relocations are required, then the portfolio should incorporate algorithms that generate replica schemas in a fast turn around time, such as GMM and LMM. If relocation is required only once for every 24 h, then the Greedy algorithm would be the best choice. On the other hand, if relocation is

required, say once every week, then perhaps Aε-Star or WA-Star can be deployed.

Identifying replica schema would necessitate content distributors or network managers to simulate the network onto their workstations, requiring the optimization of memory used for simulation. Thus, low memory utilizing heuristics such as LMM, GMM, and Greedy may be preferred over medium or high memory utilizing heuristics such as SA1, SA2, SA3, GRA, and others.

Execution time and memory utilization may be an important factor in selecting heuristic(s), but most importantly one would be interested in the solution quality of the heuristic. High solution quality heuristics, namely, Aε-Star, WA-Star, and Greedy maybe preferred over the medium quality heuristics, such as GRA, SA2, and SA3. In many scenarios the general measure of solution quality may not be an adequate criterion to select a heuristic. With the changing dynamics of the system choices may change. For instance:

1. Changes in the system storage capacity may require the content distributors or network managers to choose between either the set of heuristics that construct solution(s) incrementally (such as, Greedy, LMM, and GMM) or the set of heuristics that develop combinatorial (A-Star, A-Star based heuristics, and GRA) solution(s). Execution time and memory utilization may simply be overlooked.
2. Variation in the read/write ratio is catered well by heuristics that construct combinatorial solutions, thus contrary to the situation (change in system storage capacity) where such type of heuristics were an ill-choice, they may become an excellent choice. However, we do not have to struggle with these two polar choices. The Greedy heuristic which is an incremental method, furnished high solution quality with altering read/write ratio.

In essence, one has to make a very educated guess by incorporating every possible variation of the system parameters to select a heuristic for an effective and efficient replica placement.

References

[1] G. Abdulla, Analysis and modeling of world wide web traffic, Ph.D. Thesis, Virginia Polytechnic Institute and State University, Virginia, USA, 1998.

[2] P. Apers, Data allocation in distributed database systems, ACM Trans. Database Systems 13 (3) (1988) 263–304.

[3] M. Arlitt, T. Jin, Workload characterization of the 1998 World Cup web site, Technical Report, Hewlett Packard Laboratory, Palo Alto, CA, USA, HPL-1999-35(R.1), 1999.

[4] S. Baker, B. Moon, Scalable web server design for distributed data management, in: Proceedings of the 15th International Conference on Data Engineering, 1999, p. 86.

[5] S. Bradley, A. Hax, T. Magnanti, Applied Mathematical Programming, Addison-Wesley, Reading, MA, 1977.

[6] K. Calvert, M. Doar, E. Zegura, Modeling internet topology, IEEE Comm. Mag. 35 (6) (1997) 160–163.

[7] C. Ceri, G. Pelagatti, G. Martella, Optimal file allocation in a computer network: a solution based on knapsack problem, Comput. Networks 6 (1982) 345–357.

[8] H. Chang, R. Govindan, S. Jamin, S. Shenker, Towards capturing representative AS-Level Internet topologies, Comput. Networks J. 44 (6) (2004) 737–755.

[9] M. Charikar, S. Guha, E. Tardos, D. Shmoys, A constant-factor approximation algorithm for the K-median problem, in: Proceedings of the 31st Annual ACM Symposium on the Theory of Computation, 1999, pp. 1–10.

[10] L. Chen, H. Choi, Approximation algorithms for data distribution with load balancing of web servers, in: Proceedings of the 3rd IEEE International Conference on Cluster Computing, 2001, pp. 274–281.

[11] W. Chu, Optimal file allocation in a multiple computer system, IEEE Trans. Computers 18 (10) (1969) 885–889.

[12] S. Cook, J. Pachl, I. Pressman, The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy, Distrib. Comput. 15 (1) (2002) 57–66.

[13] B. Davison, A survey of proxy cache evaluation techniques, in: Proceedings of the 4th International Web Caching Workshop, 1999, pp. 67–77.

[14] L. Dowdy, D. Foster, Comparative models of the file allocation problem, ACM Comput. Surveys 14 (2) (1982) 287–313.

[15] K. Eswaran, Placement of records in a file and file allocation in a computer network, in: Proceedings of IFIP Congress, 1974, pp. 304–307.

[16] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, N. Young, Competitive paging algorithms, J. Algorithms 12 (4) (1991) 685–699.

[17] S. Floyd, V. Paxson, Difficulties in simulating the Internet, IEEE/ACM Trans. Networking 9 (4) (2001) 253–285.

[18] M. Garey, D. Johnson, Computers and Intractability, W.H. Freeman and Co., Murray Hills, NJ, USA, 1979.

[19] N. Gautam, An Integer programming formulation of the web server location problem, Available at ⟨http://ie.tamu.edu/people/faculty/Gautam/papers/gweb.pdf⟩.

[20] R. Gonen, D. Lehmann, Optimal solutions for multi-unit combinatorial auctions: branch and bound heuristics, in: Proceedings of the 2nd ACM Conference on Electronic Commerce, 2000, pp. 13–20.

[21] P. Habegger, H. Bieri, Modeling the topology of the Internet: an assessment, Technical Report, Institut für Informatik und angewandte Mathematik, Universität Bern, Bern, Switzerland, IM-02-2002, 2002.

[22] T. Hara, Effective replica allocation in ad hoc networks for improving data accessibility, in: Proceedings of INFOCOM, 2001, pp. 1568–1576.

[23] J. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.

[24] K. Jain, V. Vazirani, Primal-dual approximation algorithms for metric facility location and k-median problems, in: Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 2–13.

[25] S. Jamin, C. Jin, Y. Jin, D. Riaz, Y. Shavitt, L. Zhang, On the placement of Internet instrumentation, in: Proceedings of the IEEE INFOCOM, 2000, pp. 295–304.

[26] S. Jamin, C. Jin, T. Kurc, D. Raz, Y. Shavitt, Constrained mirror placement on the Internet, in: Proceedings of the IEEE INFOCOM, 2001, pp. 31–40.

[27] X. Jia, D. Li, X. Hu, D. Du, Optimal placement of web proxies for replicated web servers in the Internet, Comput. J. 44 (5) (2001) 329–339.

[28] S. Jin, A. Bestavros, Greedydual* web caching algorithm: exploiting the two sources of temporal locality in web request streams, Comput. Comm. 24 (2) (2001) 174–183.

[29] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous computing systems, IEEE Concurrency 6 (3) (1998) 42–51.

[30] J. Kangasharju, J. Roberts, K. Ross, Object replication strategies in content distribution networks, in: Proceedings of the 6th International Web Caching and Content Distribution Workshop, 2001, pp. 455–456.

[31] M. Karlsson, M. Mahalingam, Do we need replica placement algorithms in content delivery networks?, in: Proceedings of the 7th International Workshop on Web Content Caching and Distribution, 2002, pp. 516–525.

[32] S. Khan, I. Ahmad, Heuristic-based replication schemas for fast information retrevial over the Internet, in: Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004, pp. 278–283.

[33] S. Khan, I. Ahmad, A powerful direct mechanism for optimal WWW content replication, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005, p. 86.
[34] M. Korupolu, C. Plaxton, L. Rajaraman, Analysis of a local search heuristic for facility location problems, in: Proceedings of the 9th Annual ACM/SIAM Symposium on Discrete Algorithms, 1998, pp. 1–10.
[35] B. Krishnamurthy, J. Wang, On network-aware clustering of web clients, in: Proceedings of the ACM SIGCOMM, 2000, pp. 97–110.
[36] Y.-K. Kwok, K. Karlapalem, I. Ahmad, N. Pun, Design and evaluation of data allocation algorithms for distributed database systems, IEEE J. Selected Areas in Communication 14 (7) (1996) 1332–1348.
[37] B. Lee, J. Weissman, Dynamic replica management in the service grid, in: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, 2001, pp. 433–434.
[38] B. Li, M. Golin, G. Italiano, X. Deng, On the optimal placement of web proxies in the Internet, in: Proceedings of the IEEE INFOCOM, 1999, pp. 1282–1290.
[39] Looking Glass Sites Project, Available at ⟨http://www.nanog.org/lookingglass.html⟩.
[40] T. Loukopoulos, Caching and replication schemes on the Internet, Ph.D. Thesis, Hong Kong University of Science and Technology, Hong Kong, China, 2002.
[41] T. Loukopoulos, I. Ahmad, Static and adaptive distributed data replication using genetic algorithms, J. Parallel Distrib. Comput. 64 (11) (2004) 1270–1285.
[42] T. Loukopoulos, I. Ahmad, D. Papadias, An overview of data replication on the Internet, in: Proceedings of the 6th International Symposium on Parallel Architectures Algorithms, and Networks, 2002, pp. 31–36.
[43] S. Mahmoud, J. Riordon, Optimal allocation of resources in distributed information networks, ACM Trans. Database Systems 1 (1) (1976) 66–78.
[44] S. March, S. Rho, Allocating data and operations to nodes in distributed database design, IEEE Trans. Knowledge and Data Engineering 7 (2) (1995) 305–317.
[45] A. Medina, I. Matta, J. Byers, On the origin of power laws in Internet topologies, ACM Comput. Comm. Rev. 30 (2) (2000) 18–28.
[46] B. Narebdran, S. Rangarajan, S. Yajnik, Data distribution algorithms for load balancing fault-tolerant web access, in: Proceedings of the 16th Symposium on Reliable Distributed Systems, 1997, pp. 97–106.
[47] NASA Keneddy Space Center access log, Available at ⟨http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html⟩.
[48] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E.M. Nahum, Locality-aware request distribution in cluster-based network servers, in: Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, 1998, pp. 205–216.
[49] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, MA, 1984.
[50] L. Qiu, V. Padmanabhan, G. Voelker, On the placement of web server replicas, in: Proceedings of the IEEE INFOCOM, 2001, pp. 1587–1596.
[51] M. Rabinovich, Issues in web content replication, Data Engineering Bulletin 21 (4) (1998) 21–29.
[52] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, J. Kubiatowicz, Maintenance-free global storage, IEEE Internet Comput. 5 (5) (2001) 40–49.
[53] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek, Selection algorithms for replicated web servers, ACM Performance Evaluation Rev. 26 (3) (1998) 44–50.
[54] S. So, I. Ahmad, K. Karlapalem, Response time driven multimedia data objects allocation for browsing documents in distributed environments, IEEE Trans. Knowledge and Data Engineering 11 (3) (1999) 386–405.
[55] G. Srinivasan, N. Gautam, Optimal location of web servers, in: Proceedings of the Industrial Engineering Research Conference, 2002.
[56] A. Vigneron, L. Gao, M. Golin, G. Italiano, B. Li, An algorithm for finding a k-median in a directed tree, Inform. Process. Lett. 74 (2000) 81–88.
[57] B. Waxman, Routing of multipoint connections, IEEE J. Selected Areas of Communications 6 (9) (1988) 1617–1622.
[58] D. White, An extension of a greedy heuristic for the knapsack problem, European J. Oper. Res. 51 (1991) 387–399.
[59] E. Zegura, K. Calvert, M. Donahoo, A quantitative comparison of graph-based models for Internet topology, IEEE/ACM Trans. Networking 5 (6) (1997) 770–783.
[60] L. Zhuo, C. Wang, F.C.M. Lau, Load balancing in distributed web server systems with partial document replication, in: Proceedings of the 31st International Conference on Parallel Processing, 2002, pp. 305–314.
[61] G. Zipf, Human Behavior and the Principle of Least-Effort, Addison-Wesley, Reading, MA, 1949.

Samee Ullah Khan received his BS in Computer Systems Engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in May 1999, and a Ph.D. in Computer Science from the University of Texas, Arlington, TX, USA, in August 2007.
From September 2007 he will be affiliated with the Electrical and Computer Engineering Department of the Colorado State University, Fort Collins, CO, USA as a research fellow.
His research interests include designing, building, analyzing, and measuring large-scale autonomous distributed computing systems using game theoretical and algorithmic mechanism design techniques, passive optical network layouts, designing secure systems, combinatorial games, and combinatorial optimization. Dr. Khan is a member of the European Association of Theoretical Computer Science, the Game Theory Society, the IEEE Communications Society, the IEEE Computer Society, and the Society of Photo-Optical Instrumentation Engineers.

Ishfaq Ahmad received his B.Sc. degree in Electrical Engineering from the University of Engineering and Technology, Pakistan, in 1985, and his MS degree in Computer Engineering and his Ph.D. degree in Computer Science from Syracuse University, New York, USA, in 1987 and 1992, respectively. He is currently a professor of Computer Science and Engineering at the University of Texas at Arlington (UTA). Prior to joining UT Arlington, he was on the faculty of the Computer Science Department of Hong Kong University of Science and Technology (HKUST). At HKUST, he also directed the Multimedia Technology Research Center, a university-wide research center that he conceived and established with other colleagues. At UTA, he leads the Multimedia Laboratory and Institute for Research in Security (IRIS). IRIS, an inter-disciplinary research center spanning several departments, is engaged in research on futuristic technologies for homeland security and law enforcement. Professor Ahmad is known for his research contributions in parallel and distributed computing, power-aware distributed computing, multimedia systems, video compression, and security. His work in these areas is published in close to 200 technical papers in peer-reviewed journals and conferences, including three best paper awards at leading conferences and 2007 best paper award for IEEE Transactions on Circuits and Systems for Video Technology. His current research is funded by the Department of Justice (DOJ), National Science Foundation (NSF), and industry. He is an associate editor of the Journal of Parallel and Distributed Computing, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Transactions on Multimedia, IEEE Distributed Systems Online, and Cluster Computing.