GAME THEORETICAL DATA REPLICATION TECHNIQUES FOR LARGE-SCALE AUTONOMOUS DISTRIBUTED COMPUTING SYSTEMS

by

SAMEE ULLAH KHAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2007

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my advisor, Prof. Ishfaq Ahmad for his invaluable guidance through these years at UT Arlington. Without his long-term encouragement, patience, understanding, and persistent support, this dissertation would not have been completed.

I would also like to appreciate the efforts of my doctoral dissertation committee members, Drs. Che, Cook, Kung, and Lei for their precious time in reviewing this dissertation and for their valuable suggestions.

I am especially grateful to Profs. Cook and Shirazi (at the Washington State University) and Prof. Kreinovich (at UT El-Paso) for their encouragement and continued support even though they were tens of miles away – thank you.

At UT Arlington the last couple of years were made pleasant by the occasional conversations with Drs. Huber and Zaruba and Mr. Levine. I have learnt so much from you, about life, about (ir)rational thinking and almost about everything else.

I dedicate this dissertation, a half-decade effort to my parents, who made every possible and impossible effort to ensure that there was no hindrance in what I liked to pursue for education.

May 31, 2007

ABSTRACT

GAME THEORETICAL REPLICA PLACEMENT TECHNIQUES FOR LARGE-SCALE AUTONOMOUS DISTRIBUTED COMPUTING SYSTEMS

Publication No. ______

Samee Ullah Khan, PhD.

The University of Texas at Arlington, 2007

Supervising Professor: Ishfaq Ahmad

Data replication in geographically dispersed servers is an essential technique for reducing the user perceived access time in large–scale distributed computing systems. A majority of the conventional replica placement techniques lack scalability and solution quality. To counteract such issues, this thesis proposes a game theoretical replica placement framework, in which autonomous agents compete for the allocation or reallocation of replicas onto their representative servers in a self–managed fashion. Naturally, each agent's goal is to maximize its own benefit. However, the framework is designed to suppress individualism and to ensure system–wide optimization. Using this framework as an environment, several cooperative and non–cooperative low–

complexity, flexible, and scalable game theoretical replica placement techniques are proposed, analytically investigated, and experimentally evaluated. Each of these techniques supports different game theoretical (pareto–optimality, catering to agents' interests, deliberate discrimination of allocation, budget balanced, pure Nash equilibrium, and Nash equilibrium) and system (link distance, congestion control, minimization of communication cost, and memory optimization) related properties. Using a detailed test–bed involving eighty various network topologies and two real–world access logs, each game theoretical technique is also extensively compared with conventional replica placement techniques, such as, greedy heuristics, branch–and–bound techniques and genetic algorithms. The experimental study confirms that in each case the proposed techniques outperform other conventional methods. The results can be summarized in four ways: 1) The number of replicas in a system self–adjusts to reflect the ratio of the number of reads versus writes access; 2) Performance is improved by replicating objects to the servers based on the locality of reference; 3) Replica allocations are made in a fast algorithmic turn–around time; 4) The complexity of the data replication problem is decreased by multifold.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

With the exponential growth of the World Wide Web, popular web serves are required to handle enormous amount of requests from geographically and psychologically diverse users [1]. These web servers are in constant competition with their peers to provide better (faster/reliable) Internet usage. (For instance, over the number of years many commercial web hosting services such as Akamai, Exodus, etc. have gained popularity [58].) However, the Internet access simply cannot be improved by high performance web servers [100]. 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 [79]. Although it is a well-studied problem, yet its application on the Internet gave rise to new problems, e.g., where to place a cache, how to make sure cached contents are valid, and how to handle dynamic pages. Replication, in contrast, has been commonly used in distributed systems to increase availability and fault tolerance, which in turn leads to load balancing and increases client-server proximity [108]. Both techniques play complementary roles in the Internet environment [78]. 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. 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 aspect of a caching scheme has its equivalent in replicated systems, but not vice versa. For fault-tolerant and highly dependable systems, replication is essential, as demonstrated in a real world example of Ocean Store [107]. Replication can be coarse-grained (replication of an entire site or server) or fine-grained (replication of individual data items or objects). Below we detail these two popular models.

1.1 Coarse-grained replication model

Similarity to the celebrated distributed file allocation problem [109], has moved the researches to address the problem of data replication on similar lines. 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 location where it is to be replicated, it is termed as coarse-grained replication [79].

A majority of the initial work, assumed the coarse-grained replication model.

We detail some of the major works that use the coarse-grained model as follows. Authors in [75] model the Internet as a tree. Being not only unrealistic, the model also assumes 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) towards the servers. Moreover their bound of $O(N^3M^2)$ prevents them to compute real world proxy reallocation. Nevertheless work reported in [75], is the very first of its nature which deals directly with the proxy server placement. In [46], the authors used not only theoretical results, but combined it with appropriate heuristics. Their heuristic approach is so strong that it does not require the full knowledge of the network topology as assumed in many approaches [79]. They latter improved [47] their approach by introducing a refined constrained based policy. A comprehensive comparison with realistic data from Internet log files is done in [100], where the Greedy approach [100], outperforms the Dynamic Programming approach [75], Randomized [100] and Hot Spot [100]. Although replicating the entire contents of the website can reduce the hit-miss ratio by considerable amounts, yet in the context of high-performance systems, the coarse-grained replication model is a too simplistic approach [79].

1.2 Fine-grained replication model

This model allows the replication of certain objects as apposed to the entire site. This approach has many advantages, such as [100]: it saves the server memory capacity, it moves only those objects that are actually required to be reallocated, it reduces the network traffic and provides load-balancing. The generalized fine-grained replication is known to be NP-complete not only for the general graphs [78], but also for the partitioned graphs [52]. We detail some of the major works that use the fine-grained model as follows. Authors in [78] analyzed both static (such as a modified Greedy based approach [83] and an Evolutionary method based on Genetic algorithms [44]) and adaptive (such as a self-configured Genetic approach) replication techniques. Experimental results revealed that the static Genetic approach outperformed on every occasion. The work was further extended [77] with comparisons to Linear Programming [12] and Linear Integer Programming [36]. In [110], 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 [77].

This text focuses on the algorithms for the placement of replicas. Replica placement techniques 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 additional copies, against the cost of storing and updating the replicas. In general, clients experience reduced access latencies provided that data is 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 [78]. With both 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 [58]. Therefore, efficient and effective replication schemas strongly depend on how many replicas to be placed in the system, and more importantly where [79].

Myriad theoretical approaches are proposed that we classify (and also describe a few seminal works) into the following six categories:

1.2.1 Facility Location

The facility location problem can be defined as: *"Find a location that minimizes a weighted sum of distances to each of several locations"*.

The generalized facility location problem is NP-complete [35]. The only known work on data replication with similar characteristics as that of the facility location problem is reported in [46]. 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 [77].

1.2.2 File Allocation

File allocation has been a popular line of research in: distributed computing [83], distributed databases [5], multimedia databases [109], paging algorithms [33], and video server systems [109]. The generalized file allocation problem for multiple objects [21] has been proven to be NP–complete [31]. We can formally state the file allocation problem with context to the replication problem as [79]: *"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 [79]. There the authors provide a guaranteed optimal result, but since the assumption is on unlimited capacity, the result is of little practical use [54]. A rather old but a comprehensive survey on file allocation can be found in [30].

1.2.3 Minimum k-Median

The celebrated NP-complete minimum k–median problem is formally defined as follows [79]: *"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(s)".*

A lot of work has been done on k–median and its variants, e.g., [45], [64]. In [75] 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 [115]. Both the results have significant theoretical contributions 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 seen in [49]. To compliment these approaches, a more generalized solution was presented in [100]. There the authors compared various placement techniques and proposed a greedy algorithm that outperformed other techniques including the work reported in [75]. Most of the results in this category have already been discussed while describing the coarse-grained model.

1.2.4 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 [79], yet it requires a lot of maneuverability in terms of choosing the optimization function [18]. In [52], 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 very conservative assumptions (such as, fixed location of the original server, access patterns are to be known before hand, no network failures, etc.) as addressed in [47] and [73]; therefore, they can not handle the dynamics of the system [79].

1.2.5 Bin Packing

Widely studied in the field of on-line algorithms, the bin packing problem is known to be NP-hard [35]. We can formally state the problem with context to replication as [79]: *"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 [89], where the authors formulate the problem over a cluster of web servers to reduce the server loads, by incorporating the usage of dummy replicas. However, their approach performs well only when the network under consideration is small. Most recently a more flexible and general approach was undertaken in [57]. There the authors performed extensive experimental comparisons using various network topologies and real access logs.

1.2.6 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 [9]. Many researchers [89] 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 [97]. 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 can be to reduce the communication cost or latency etc. If no such replica is unassigned the problem is reduced to 0-1 knapsack [35]. Due to the close resemblance of the knapsack problem to the bin packing problem, it is widely studied by researchers in the filed of Operations Research [119], Game Theory [37] and Approximation Algorithms [123]. Authors in [20] have proved that minimizing the maximum load over all the web server nodes is NP-complete. If the constraint of load balancing is removed, the problem of minimizing the communication cost still remains

Table 1.1: The major work reported in the field of replica placements.

Category	Work	Problem definition related assumptions					Experimental related Information	
		Topology assumed	Access		Objects	Storage	Access patterns	Topologies
			Reads	Writes	Multiple/Single	Constraints		
1	[46]	General graphs	Yes	No	Single	No	Synthetic (Zipf)	AS, Transit-Stub
2	[5]	General, fork graphs	Yes	Yes	Single	No	N/A	N/A
	[21]	Fully connected graphs	Yes	Yes	Single	No	N/A	N/A
	[31]	General graphs	Yes	No	Single	No	Statistical	Flat
	[33]	Uniform cost graphs	Yes	No	Multiple	No	N/A	N/A
	[70]	General graphs	Yes	Yes	Multiple	Yes	Statistical	Flat
	[83]	General graphs	Yes	Yes	Single	No	N/A	N/A
	[109]	General graphs	Yes	Yes	Multiple	Yes	Statistical	Flat
3	[45]	Linear, general graphs	Yes	No	Single	No	N/A	N/A
	[52]	General graphs	Yes	No	Multiple	No	Synthetic (Zipf)	AS
	[49]	Trees	Yes	No	Single	No	Statistical	Trees
	[64]	General graphs	Yes	No	Multiple	Yes	N/A	N/A
	[75]	Trees	Yes	No	Single	No	Statistical	Trees
	[115]	Trees, general graphs	Yes	No	Multiple	No	Statistical	Tress, Flat
4	[18]	Linear, general graphs	Yes	No	Multiple	No	N/A	N/A
	[47]	General graphs	Yes	Yes	Multiple	No	Synthetic (Zipf)	AS, Transit-Stub
	[73]	General graphs	Yes	Yes	Multiple	Yes	Real-time	Flat
	[100]	Trees, general graphs	Yes	Yes	Multiple	Yes	Access logs	Tress, Flat
5	[57]	General graphs	Yes	Yes	Multiple	Yes	Access logs	Flat
	[89]	Linear graphs	Yes	Yes	Multiple	Yes	Statistical	Flat
6	[16]	General graphs	Yes	Yes	Single	No	N/A	N/A
	[26]	Linear, general graphs	Yes	Yes	Single	No	N/A	N/A
	[78]	General graphs	Yes	Yes	Multiple	Yes	Synthetic (Zipf)	Flat
	[108]	General graphs	Yes	Yes	Multiple	No	N/A	N/A

NP-hard [124]. A rather different approach [26] 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 of the significance work in this line of pursuit is reported in [16], [78] and [108].

A number of bibliographies and reading materials for web caching are also available online, e.g., [26]. An overview of replication and its challenges are provided in [79] and [100], respectively. Table 1.1 provides an overview of the major work reported categorized into the six major theoretical oracles.

Our aim here is to propose, design and analyze efficient and effective fine-grained replica placement techniques. Naturally, a stringent conventional benchmark needs to be set before one can pursue original thoughts. For this purpose, we first propose an infrastructure that close mimics the Internet (an example of large-scale distributed computing system) in topology and traffic. This infrastructure is benchmarked with ten static heuristic based techniques with various problem parameters. This rigorously benchmarked data is recorded for comparisons with originally conceived, novel, game theoretical replica placement techniques.

Our aim (through this study) is to show that game theoretical techniques are per se better than conventional centralized techniques because of their flexibility in design, distributed control, scalability, and various levels of optimality.

In the remainder of this document, after a brief survey on the current state of the art replica placement techniques in Chapter 2, we will introduce the replica placement problem, the underlying system assumptions, parameters, and an experimental infrastructure that closely mimics the Internet in topology and traffic in Chapter 3. A set of ten static heuristics will be put to test to benchmark conventional replica placement techniques in Chapter 4. In Chapter 5, we provide back ground information on game theory and design an incentive compatible game theoretical replica placement technique. This technique is further extended to capture a few important properties of pareto optimality, utility maximization, deliberate discrimination, budget balance, and a cooperative technique in Chapters 6, 7, 8, 9, and 10, respectively. Finally, Chapter 11 provides conceptual views and discusses extension to the work presented here.

CHAPTER 2

STATE OF THE ART REPLICA PLACEMENT TECHNIQUES

The replica placement problem is an extension of the classical file allocation problem (FAP). Chu [21] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [15] extended this work by distinguishing between updates and read file requests. Eswaran [31] proved that Casey's formulation was NP complete. In [82] Mahmoud *et al.* provide an iterative approach that achieves good solution quality when solving the FAP for infinite server capacities. A complete although old survey on the FAP can be found in [30]. Apers in [5] considered the data allocation problem (DAP) in distributed databases where the query execution strategy influences allocation decisions. In [70] the authors proposed several algorithms to solve the data allocation problem in distributed multimedia databases (without replication), also called as video allocation problem (VAP). Replication algorithms fall into the following three categories:

1. The problem definition does not cater for the user accesses.
2. The problem definition only accounts for read access.
3. The problem definition considers both read and write access including consistency requirements.

These categories are further classified into four categories according to whether

Table 2.1: Summary of related work.

Category	Number of objects	Storage constraints	References
Category 1: *No object access.*	Single object	No storage constraint	[46].
		Storage constraint	–
		No storage constraint	[42], [47].
	Multiple objects	Storage constraint	–
Category 2: *Read accesses only.*	Single object	No storage constraint	[30], [42], [47], [55], [64], [67], [75], [100].
		Storage constraint	[10], [23], [54], [56], [69].
		No storage constraint	[9], [52], [56], [74], [82], [113].
	Multiple objects	Storage constraint	[21], [89].
Category 3: *Read and write accesses.*	Single object	No storage constraint	[5], [21], [26], [82], [120], [121].
		Storage constraint	[19], [53], [65], [80], [82], [110].
		No storage constraint	[43], [67], [100], [101].
	Multiple objects	Storage constraint	[8], [1], [57], [58], [78], [82], [87].

a problem definition takes into account single or multiple objects, and whether it considers storage costs. Table 2.1 shows the categorized outline of the previous work reported.

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

Most of the research papers tackle the problem definition of category 2. They are applicable to read-only and read-mostly workloads. In particular this category fits well in the context of content distribution networks (CDNs). Problem definitions [42], [54], [81] and [111] have all been used in CDNs. The two main differences between them are whether they consider single or multiple objects, and whether they consider storage costs or not. The cost function in [82] also captures the impact of allocating large objects and could possible be used when the object size is highly variable. In [30] the authors tackled a similar problem – the proxy cache placement problem. The performance metric used there was the distance parameter, which consisted of the distance between the client and the cache, plus the distance between the client and the node for all cache misses. It is to be noted that in CDN, the distance is measured between the cache and the closest node that has a copy of the object.

The storage constraint is important since it can be used in order to minimize the amount of changes to the previous replica placements. As far as we know only the works reported in [57] and [78] have evaluated the benefits of taking storage costs into consideration. Although there are research papers which consider storage constraints in their problem definition, yet they never evaluate this constraint (e.g. see [31], [47], [75] and [100]).

Considering the impacts of writes, in addition to that of reads, is important, if content providers and applications are able to modify documents. This is the main characteristic of category 3. Some research papers in this category also incorporate consistency protocols – in many different ways. For most of them, the cost is the number of writes times the distance between the client and the closest node that has the object, plus the cost of distributing these updates to the other replicas of the object. In [46], [52], [55] and [75] the updates are distributed in the system using a minimum spanning tree. In [47] and [100] one update message is sent from the writer to each copy, while in [57] and [78] a generalized update mechanism is employed. There a broadcast model is proposed in which any user can update a copy. Next, a message is sent to the primary (original) copy holder server which broadcasts it to the rest of the replicas. This approach is shown to have lower complexity than any of the above mentioned techniques. In [67] and [102], it is not specified how updates are propagated. The other main difference among the above definitions is that [52], [55], [57], [75], [78] and [113] minimize the maximum link congestion, while the rest minimize the average client access latency or other client perceived costs. Minimizing the link congestion would be useful, if bandwidth is scare.

Some on-going work is related to dynamic replication of objects in distributed systems when the read-write patterns are not known *apriori*. Awerbuch's *et al.* work in [7] is significant from a theoretical point of view, but the adopted strategy for commuting updates (object replicas are first deleted), can prove difficult to implement in a real-life environment. In [119] Wolfson *et al.* proposed an algorithm that leads to optimal single file replication in the case of a tree network. The performance of the scheme for general network topologies is not clear though. Dynamic replication protocols were also considered under the Internet environment. Heddaya *et al.* [43] proposed protocols that load balance the workload among replicas. In [100], Rabinovich *et al.* proposed a protocol for dynamically replicating the contents of an Internet service provider in order to improve client-server proximity without overloading any of the servers. However updates were not considered.

Recently, game theory has emerged as a popular tool to tackle optimization problems especially in the field of distributed computing. However, in the context of data replication it has not received much attention. We briefly elaborate three pinoring works that directly or indirectly deal with the replica placement problem using game theoretical techniques. The first work [22] is mainly on caching and uses an empirical model to derive Nash equilibrium. The second work [58] focuses on mechanism design issues and derives an incentive compatible auction for replicating data on the Web. The third work [72] deals with identifying Nash strategies derived from synthetic utility functions. Our work differs from all the game theoretical techniques in: 1) identifying a non-cooperative non-priced based replica allocation method to tackle the data replication problem, 2) using game theoretical techniques to study an environment where the agents behave in a selfish manner, 3) deriving pure Nash equilibrium and pure strategies for the agents, 4) performing extensive experimental comparisons with a number of conventional techniques using an experimental setup that is mimicking the Web in its infrastructure and access patterns.

CHAPTER 3

THE REPLICA PLACEMENT PROBLEM

3.1 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. Below we elucidate the few system related assumptions.

1. Each site is assigned a unique site identifier. There a total of M sites in the system and S^i ($1 \leq i \leq M$) 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 ($1 \leq k \leq N$) 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 [58]. 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 [77]:

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 (*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.

3.2 Replica Placement Problem Formulation

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 $1 \leq i \leq M$. 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 [46], [57], [78], [100]. Let there be N objects, each identifiable by a unique name O_k and size in simple data unites o_k where $1 \leq k \leq N$. 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 study we use analytical data that will enable us to effectively predict the time interval t (in the subsequent text it will be described in detail how this relocation period can be identified).

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 k-th 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 k-th 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 (NTC) or the total object transfer cost (OTC). (We will use the terms RC, NTC and OTC interchangeably. They all point to the same measure.) The communication cost of the control messages has minor impact to the overall performance of the system [79], 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:

$$R_k^i = r_k^i o_k c(i, NN_k^i)\,, \tag{3.1}$$

where $NN_k^i = \{Site\ j \mid j \in R_k \wedge \min c(I,j)\}$. 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:

$$W_k^i = w_k^i o_k (c(i, P_k) + \sum_{\forall (j \in R_k), j \neq i} c(P_k, j))\,. \tag{3.2}$$

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:

$$C_{overall} = \sum_{i=1}^{M} \sum_{k=1}^{N} (R_k^i + W_k^i)\,. \tag{3.3}$$

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. Equation 3 is now refined to:

$$X = \sum_{i=1}^{M} \sum_{k=1}^{N} (1 - X_{ik})[r_k^i o_k \min\{c(i,j) \mid X_{jk} = 1\} + w_k^i o_k c(i, P_k)] + X_{ik}(\sum_{x=1}^{M} w_k^x \tag{3.4}$$

.

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:

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

and subject to the primary copies policy:

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

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 replica placement problem is proven to be NP-complete [78].

3.3 The Simulation Model

Below we will describe for the curious readers the simulation model that will be used in the subsequent text. All the proposed techniques (heuristics and game theoretical) are extensively compared using an experimental setup that closely mimics the Internet in its infrastructure and user access patterns. GT-ITM [14] and Inet [17] 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 web server and NASA Kennedy Space Center web server. The proposed techniques 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.

3.3.1 Performance Metric

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

3.3.2 Network Topologies

To establish diversity, the network connectively has to be changed considerably. We used four types of network topologies, which we explain below. (All in all we employed 80 various topologies.)

3.3.2.1 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.

3.3.2.1.1 Pure Random Model

A random graph $G(M,P(\text{edge} = p))$ with $0 \leq p \leq 1$ 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 [41]. The 5 pure random topologies were obtained using GT-ITM [122] topology generator with $p = \{0.4, 0.5, 0.6, 0.7, 0.8\}$.

3.3.2.1.2 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 [117]:

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

where L is the maximum distance between any two nodes and α, $\beta \in (0,1]$. 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 connectively of the graph. The smaller the value of α the larger is the number of short edges [41]. The 12 Waxman topologies were obtained using the GT-ITM [122] topology generator with values of $\alpha = \{0.1, 0.15, 0.2, 0.25\}$ and $\beta = \{0.2, 0.3, 0.4\}$.

3.3.2.2 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 serves is reversed mapped to the communication cost of transmitting a 1kB 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 1Mbps, then it means that the cost to communication 1kB of data between the two nodes is equivalent to 10-km/(2.8×10^8 m/s) + 1kB/(1Mbps) = 8.03 ms, and the cost would simply be 0.00803.

3.3.2.2.1 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 kilometers) 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 $d = \{5, 10, 15, 20\}$.

3.3.2.2.2 Fully Connected Random Graphs

This method is similar to the one used for generating random graphs except that now we do not require the node degree since the entire graph is fully connected. The 5 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 5 elements $d_1,\dots d_5$. 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 $d_1 = [1,10]$.

3.3.2.2.3 Fully Connected Uniform Graphs

This method is similar to the one described for generating fully connected random graphs except that the bandwidth and link distance are chosen uniformly and not randomly. The 5 random topologies were obtained using, $b = [1, 100]$ and $d = \{d_1=[1,10], d_2=[1,20], d_3=[1,50], d_4=[10,20], d_5=[20,50]\}$.

3.3.2.2.4 Fully Connected Lognormal Graphs

This method is similar to the one described for generating fully connected random graphs 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 [34] for an insight on the lognormal distribution functions.) The 9 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.

3.3.2.3 Power-Law Model

The power-law model [86] takes its inspiration from the Zipf law [123], and incorporates rank, out-degree and eigen exponents. We used Inet [17] 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-one nodes. Briefly, Inet starts form the total number of desired nodes and computes the number of months t it would take to grown 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 $k = \{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$.

3.3.2.4 Hierarchical Transit-Stub Model

The Internet model at the autonomous system (AS) level can also be captured by using a hierarchical model. Authors in [122] 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 [122] models a three-level hierarchy corresponding to transit domains, stub domain, and LANs attached to stub domains [41]. 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.

3.3.3 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 [6] and NASA Kennedy Space Center website [90]. 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.

3.3.3.1 Soccer World Cup Access Logs

We used eighty eight days of the Soccer World Cup 1998 access logs, *i.e.*, the (24 hours) 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

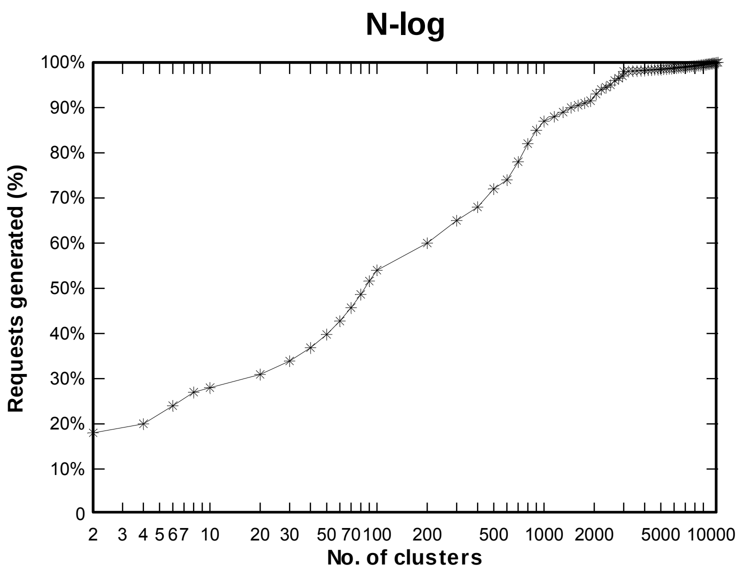

Figure 3.1: Number of requests generated by the Web clusters defined by IP address prefixes.

(maximum experimental setup). 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×Total Primary Object Sizes*. The variances in the object size collected from the access were used 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 were deemed necessary we will refer to Soccer World Cup access logs as W-log.

3.3.3.2 NASA Kennedy Space Center Access Log

We used thirty one days of the NASA Kennedy Space Center access logs, *i.e.*, the (24 hours) 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 [68] to map users onto the nodes of the topologies. The method described in [68] 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 [76] 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 Figure 3.1) 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. Notice that assigning a cluster, say C^i, to a node S^i in the network topology means that the 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×Total Primary Object Sizes*. The variances in the object size collected from the access were used 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 where deemed necessary will refer to NASA Kennedy Space Center access logs as N-log.

3.3.4 Further Clarifications on the Simulation 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 in the range of 3718 and 3728, *i.e.*, there were approximately 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 two 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 $M = 3728$.

3.3.5 The Determination of the Relocation Period

As noted previously, 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

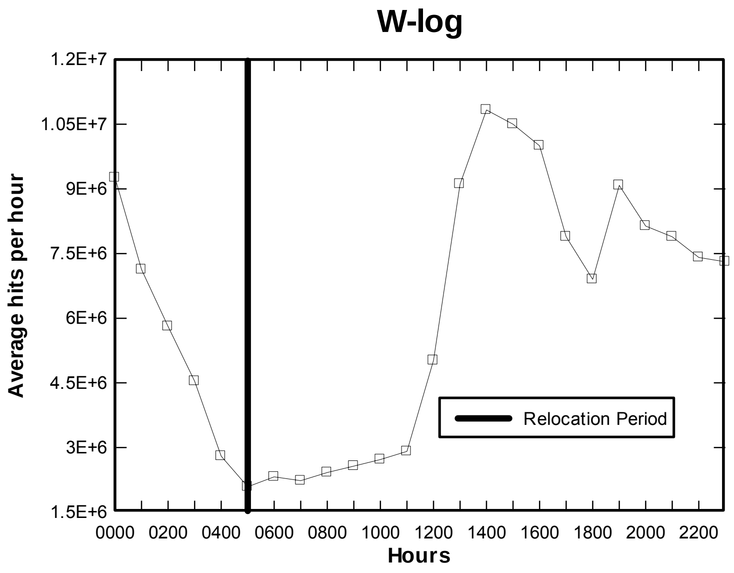

Figure 3.2: User access pattern extracted from W-log.

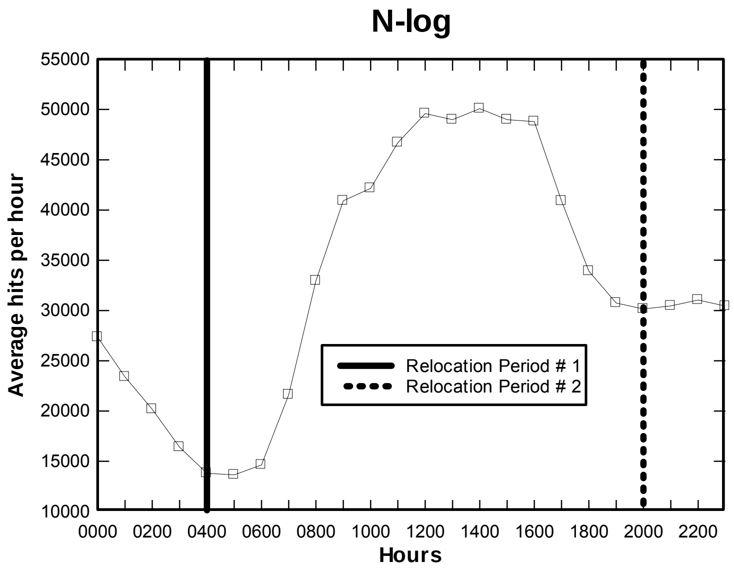

Figure 3.3: User access pattern extracted from N-log.

the replica placement techniques depends on the past trends of the user access patterns. Figures 3.2 and 3.3 show the average (over the entire access log) user access patterns extracted from the W-log and N-log. From Figure 3.2 we can clearly see that the Soccer World Cup 1998 website incurred soaring and stumpy traffic at various intervals during the 24-hour 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 hrs. 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 hrs. In our experiments we did not use this time period since the volume of traffic at 1800 hrs. 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 hrs and at 2000 hrs. This is denoted by the two vertical lines in Figure 3.3. Therefore, for the N-log a replica placement algorithm could be initiated twice daily: 1) at 0400 hrs and 2) at 2000 hrs. The time interval t for 0400 hrs would be t = (2000-0400) = 6 hours and for 2000 hrs t = (0400-2200) = 18 hours. For the W-log a replica placement algorithm could be initiated once daily at 0500 hrs. The time interval t for 0500 hrs would be t = (0500-0500) = 24 hours.

CHAPTER 4

APPROACHING THE REPLICA PLACEMENT PROBLEM USING CONVENTIONAL HEURISTICS

The unified cost model that captures the minimization of the total object transfer cost in the system, which in turn leads to the effective utilization of server side space, replica consistency, fault-tolerance, and load balancing, is used to identify replica placements using heuristics. The heuristic techniques studied include six A-Star based algorithms, two bin packing algorithms, a greedy and a genetic algorithm.

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 our experimental results, we make suggestive uses of the studied heuristics, and identify algorithm(s) that produce optimal and suboptimal replica placements. The main objective of performing such a study is to provide a benchmark which is thorough and complete in all respects. This benchmark (of conventional heuristics) will be used to earmark the performance of game theoretical techniques which is the focus here.

4.1 A-Star Based Technique (DRPA-Star)

A-Star is a best-first search algorithm based on a μ-ary tree [99]. 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 very common technique used for memory bounded A-Star type algorithms (for further details on memory bounded A-Star techniques see [51]). 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 replica placement problem 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 replication cost. Thus,

```
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 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).

Figure 4.1: Pseudo-code for DRPA-Star.

we can define the minimum of such a cost as a set: $T = min_{0\leq j\leq N-1}(t(O_k, S^i)), \forall O_k \in U$. For a node *n*, let *mmk*(*n*) define the maximum element of set *T* (the max-min replication cost). *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 Figure 4.1.

Lemma 4.1: *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 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 $h(x) \leq h(goal)$. 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 OAS's and potentially n OAS with cascaded XOR of assignments would eventually result in the solution for the DRP. ■

Lemma 4.2: *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 replication cost for u and v respectively, and $c_u \leq c_v$. 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 $h_u \geq h_v$. Since by definition: $h_u \leq h_v$, 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 4.1. ■

Lemma 4.3: *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 OAS's are identified, and the $\log(P)$ factor identifies the growth of the search tree. Now if 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 [99]. 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: $OPT_{\text{cost}}-A_{\text{cost}} \leq O(\log(OPT_{\text{cost}}))$, where A_{cost} is the admissible cost. We can thus say that $P \leq Mhd \leq dM^2$. 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

Arguably DRPA-Star is an algorithm that goes about the optimization mission too seriously. Therefore, we are interested in identifying ways to reach to a quick solution though perhaps sub-optimal. 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, a weighted evaluation function is recommended: $f(n)=(1-w)\times g(n)+w\times h(n)$. Analytical results [99] have shown that w can have three values, *i.e.*, 0, ½, 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 4.4: *WA-Star identifies a solution within a range of* $1+\boldsymbol{\varepsilon}$ *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 ($g^*(n')$) [99]. Therefore, we have:

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

$\leq \quad 1+\varepsilon.$ ■

4.2.2 Aε-Star

Perhaps a natural way to speedup any searching technique is to focus on a solution space that some how can guarantee that search in that particular space would not deviate from the optimal solution by a factor of, say ε. Keeping this 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+ε. That is, we can say that:

$$\text{FOCAL} = \{n \mid f(n) \leq (1+\varepsilon) \min_{n' \in \text{OPEN}} f(n')\}.$$

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 ε = 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+ε of DRPA-Star are expanded.

Lemma 4.5: *Aε-Star identifies a solution within a range of* 1+ **ε** *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:

$$
\begin{array}{lcll}
f(n') & \leq & f(n) & \text{(Since } h \text{ is admissible and OPEN is } f \text{ ordered.)} \\
f(t) & \leq & f(n')(1+\varepsilon) & \text{(Since } t \text{ is chosen from FOCAL.)} \\
 & \leq & f(n)(1+\varepsilon) & \\
 & \leq & 1+\varepsilon. & \blacksquare
\end{array}
$$

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, 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 [47]. 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

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 Heuristics (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 replication cost 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 4.6 ([57])**:** *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 replication costs associated with an object. The minimum of such cost as a set $T = min_{0 \leq j \leq N-1}(k(O_k,S^i), \forall O_k \in U$. If during the assignment, the minimum replication cost 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 replication cost that would occur if object O_k is replicated to a site S^i, *i.e.*, Global Min-Min (GMM).

Lemma 4.7 ([57])**:** *GMM requires* $O(M^2N^2(\log N))$ *time and* $\Omega(MN)$ *space.* ■

4.5 Greedy Based Heuristic (Greedy)

The Greedy algorithm reported in [100] 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 replication cost. 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 [100].

Lemma 4.8 ([100]**):** *Greedy requires* $O(M^2N)$ *running time.* ■

4.6 Genetic Algorithm Based Heuristic (GRA)

In [78] 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 article. 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 *k*-th bit of the *i*-th gene denotes that the *i*-th site holds a replica of the *k*-th 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 [78] for further details about the GRA method.

Lemma 4.9 ([78]**):** *GRA requires* $O(N_gN_pM^2N+N_pMN^2)$ *running time.* ■

4.7 Comparative Analysis of Proposed Heuristics

We record the performance of the heuristics using the two access logs and 80 topologies. The plots shown 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. Below we detail our findings.

4.7.1 Impact of Change in the Number of Sites and Objects

We study the behavior of the placement techniques when the number of sites increases (Figures 4.2-4.5), by setting the number of objects to 25,000; while in Figures 4.6-4.7, 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 (Figure

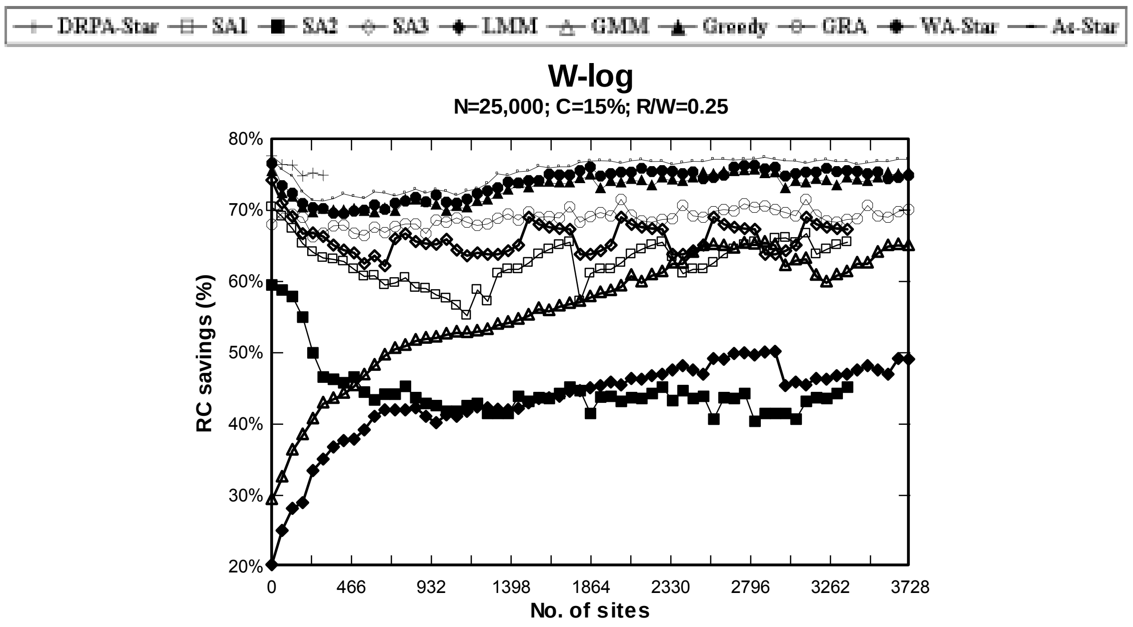

Figure 4.2: RC versus number of sites (W-log).

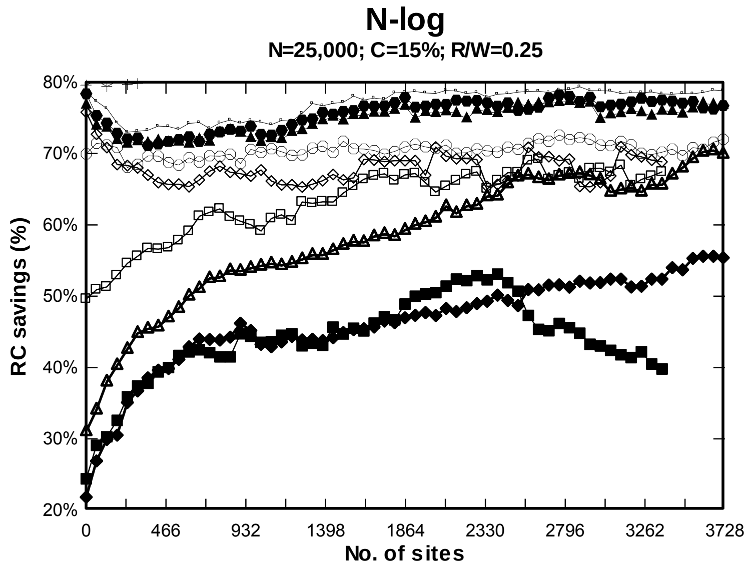

Figure 4.3: RC versus number of sites (N-log).

4.2). 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

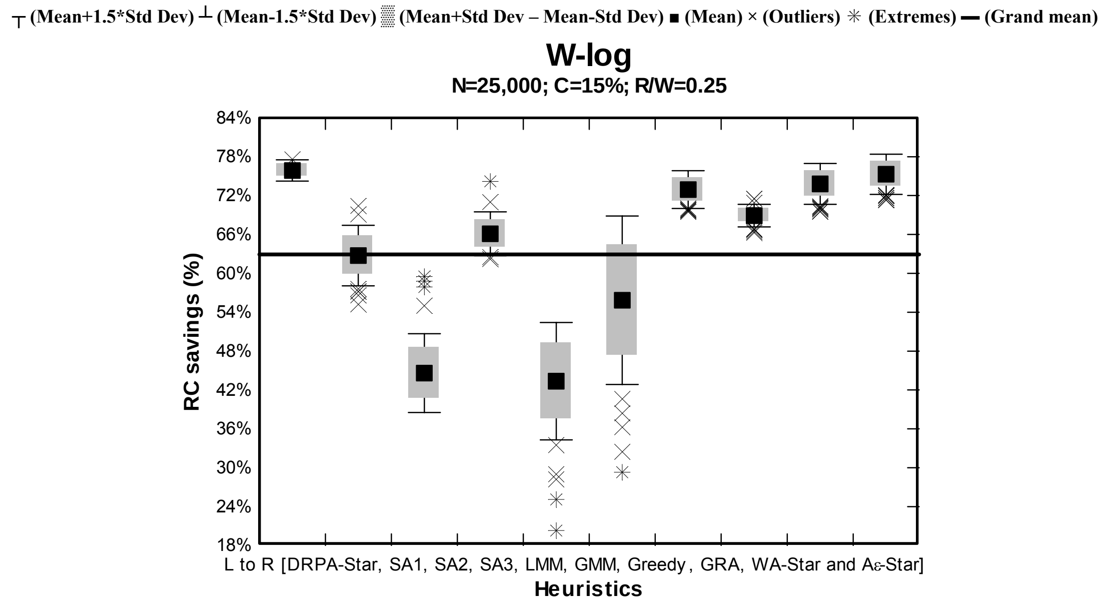

Figure 4.4: RC versus relative performance of heuristics (number of sites; W-log).

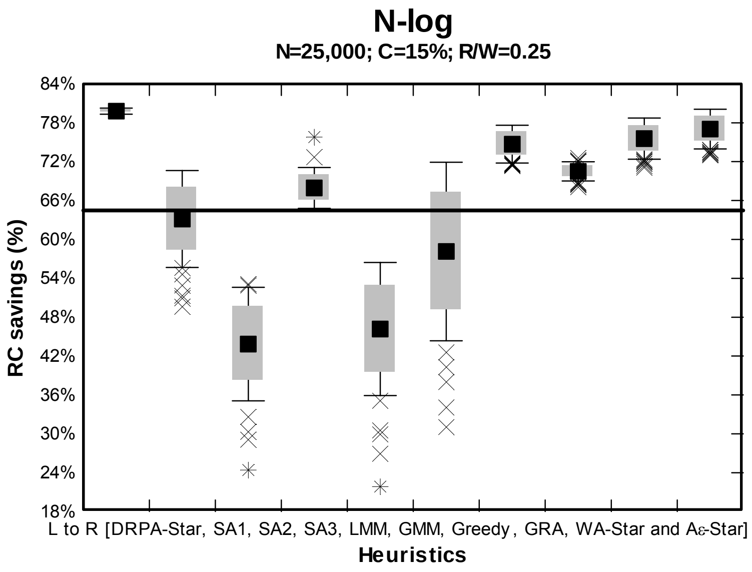

Figure 4.5: RC versus relative performance of heuristics (number of sites; N-log).

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

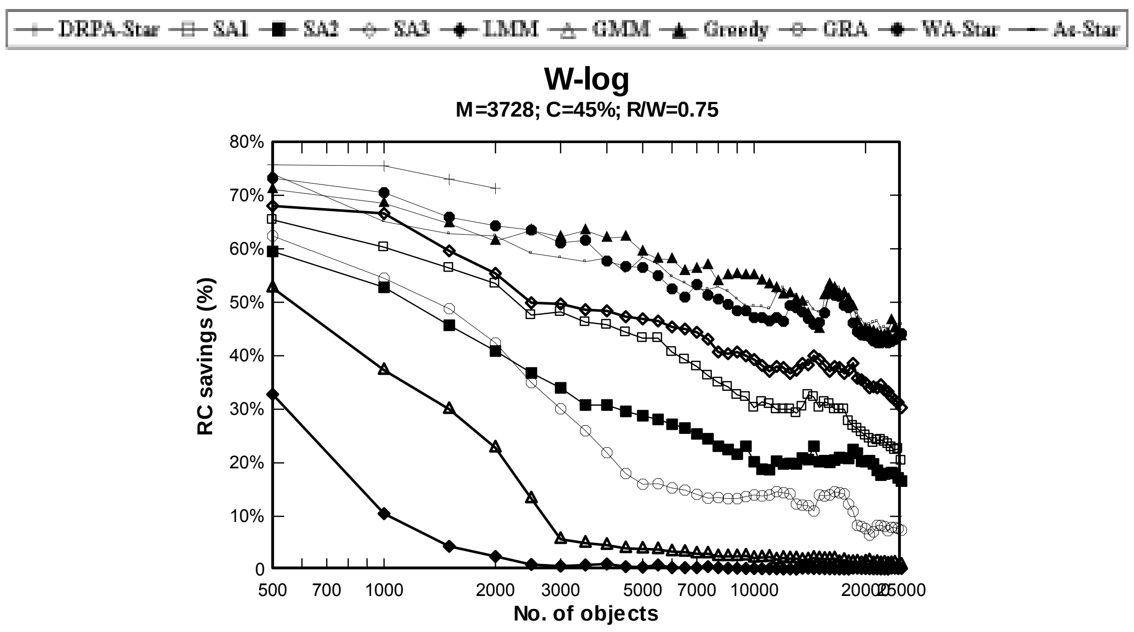

Figure 4.6: RC versus number of objects (W-log).

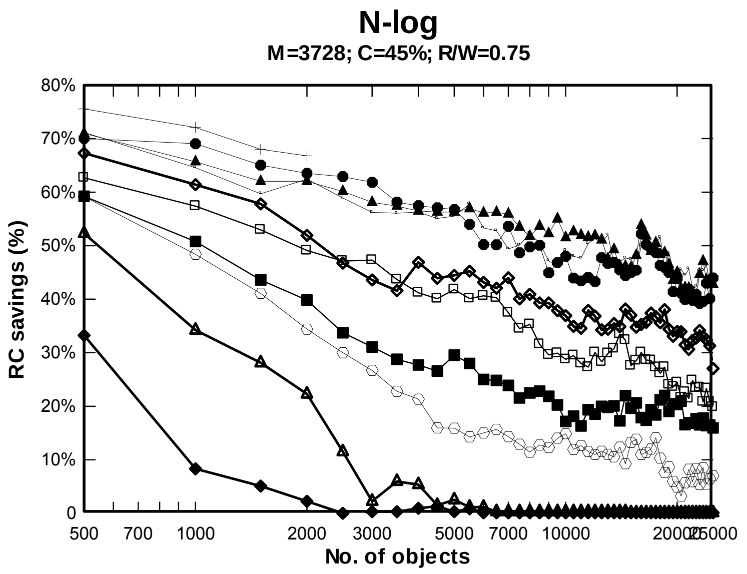

Figure 4.7: RC versus number of objects (N-log).

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 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 [78]. The observation made by using the N-log (Figure 4.3) 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 Figure 4.4 and Figure 4.5, 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 (×) represents the outliers, and the asterisks (✳) denotes the extremes. We limit the outliers and extremes to 2 and 3

┬ (Mean+1.5*Std Dev) ┴ (Mean-1.5*Std Dev) ▒ (Mean+Std Dev – Mean-Std Dev) ■ (Mean) × (Outliers) ✳ (Extremes) — (Grand mean)

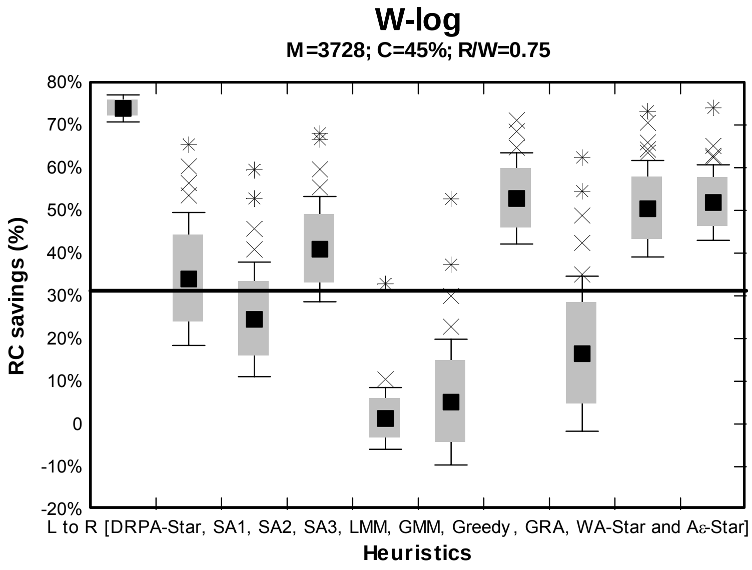

Figure 4.8: RC versus relative performance of heuristics (number of objects; W-log).

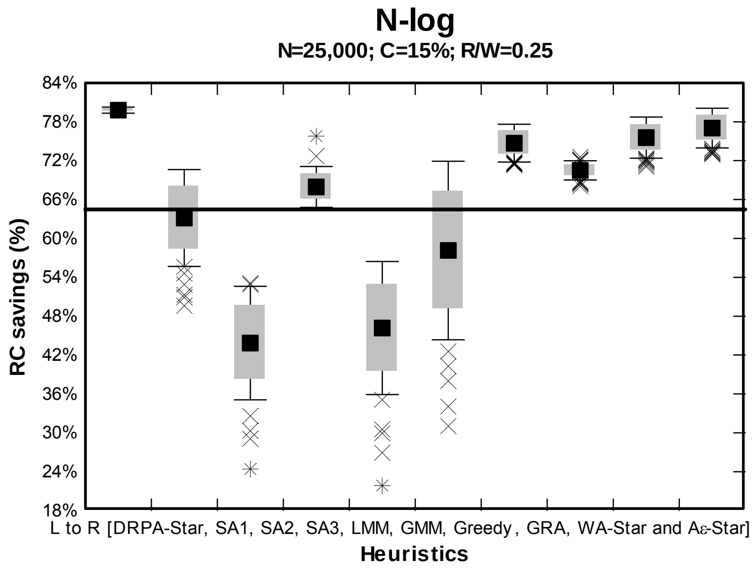

Figure 4.9: RC versus relative performance of heuristics (number of objects; N-log).

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)

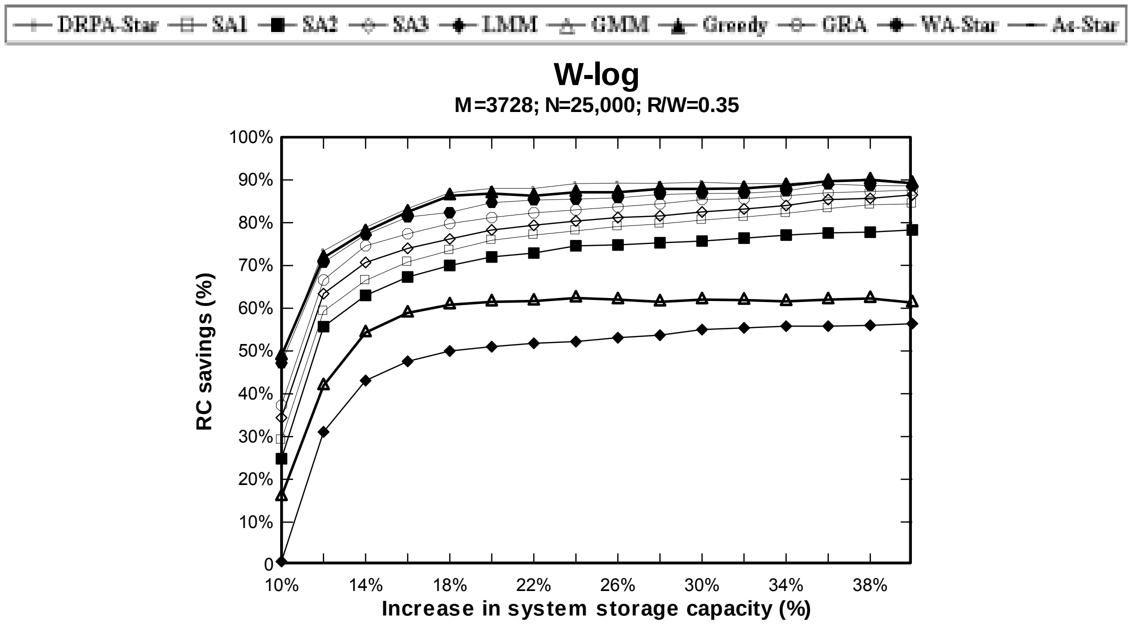

Figure 4.10: RC versus system capacity (W-log).

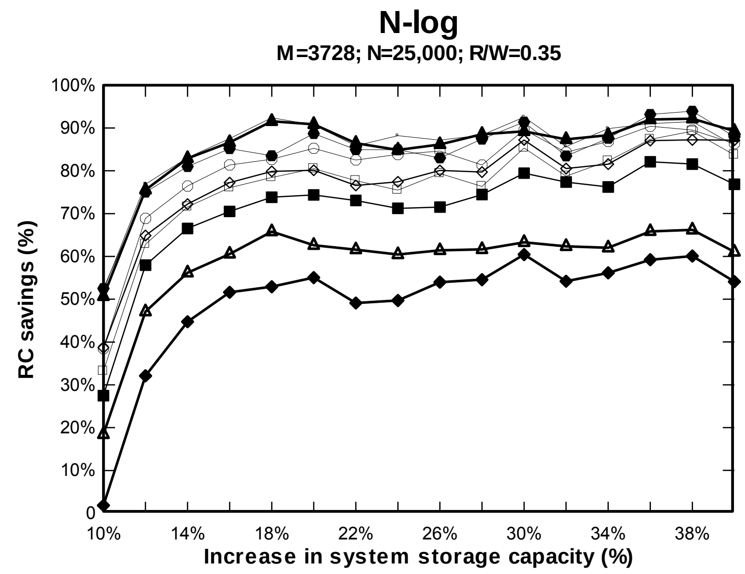

Figure 4.11: RC versus system capacity (N-log).

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 Figures 4.6-4.7. 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 (Figure 4.6). 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 [78]. But there the authors had used a uniformly distributed link cost topology, and their traffic was based on the Zipf distribution [123]. While the traffic access logs of the World Cup 1998 are more or less double-Pareto in nature [58]. 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 (Figure 4.7), the relative performance of the algorithms dropped further, this is due to the fact 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

┬ (Mean+1.5*Std Dev) ┴ (Mean-1.5*Std Dev) ▒ (Mean+Std Dev – Mean-Std Dev) ■ (Mean) × (Outliers) ✳ (Extremes) ▬ (Grand mean)

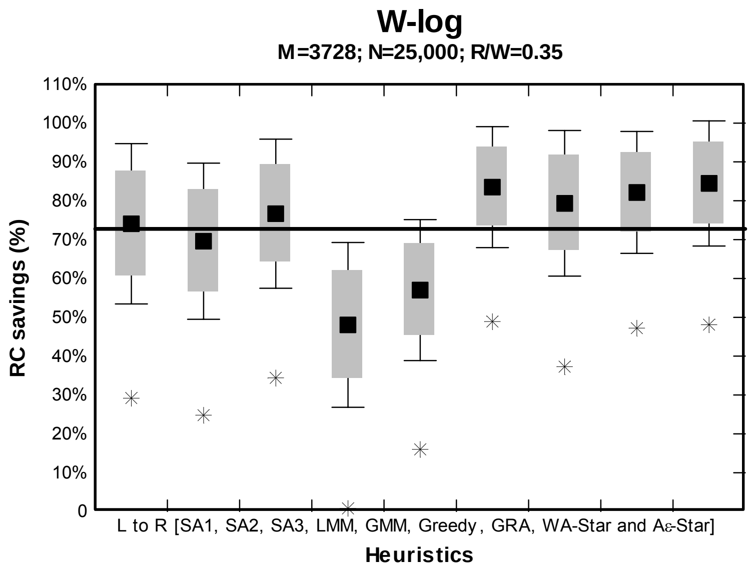

Figure 4.12: RC versus relative performance of heuristics (system capacity; W-log).

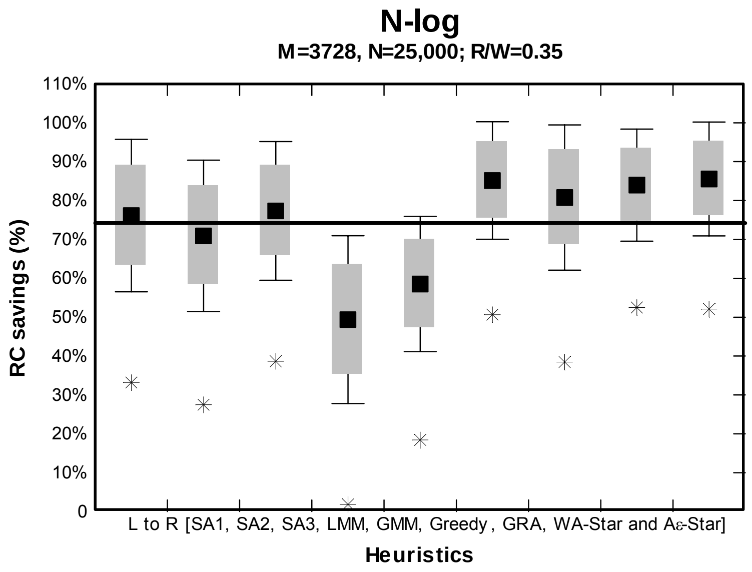

Figure 4.13: RC versus relative performance of heuristics (system capacity; N-log).

objects. However, this drop in RC savings is not more that 5%-10% compared to that of the results obtained from W-log. To better understand this phenomenon, readers are encouraged to examine the relative trends observable from Figures 4.8-4.9. 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.

4.7.2 Impact of Change in 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 (Figure 4.10). 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

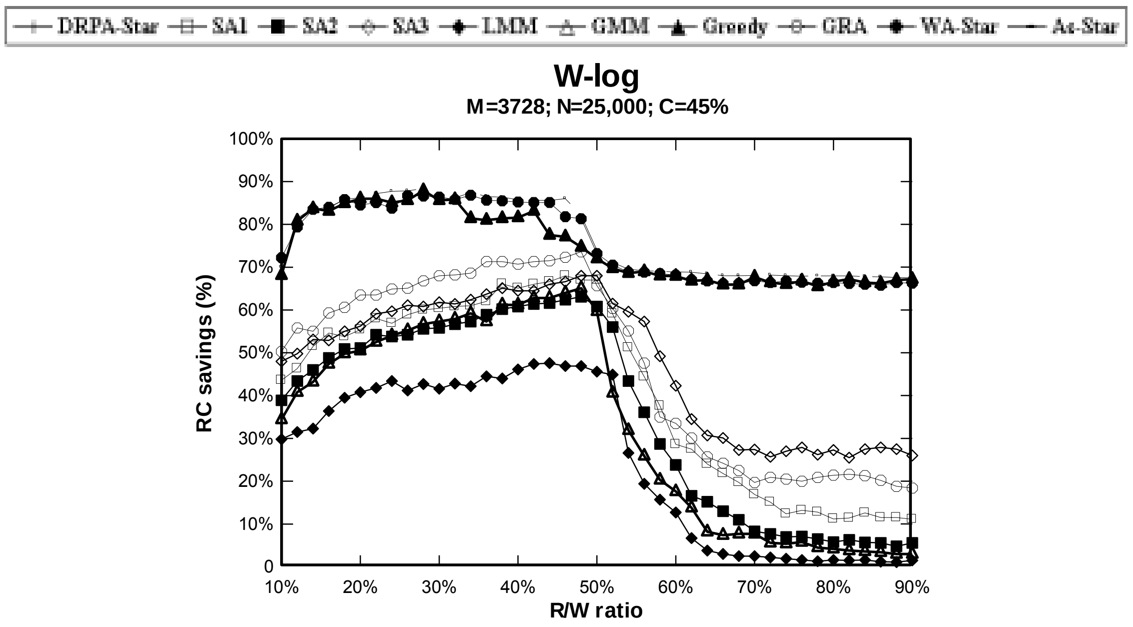

Figure 4.14: RC versus R/W ratio (W-log).

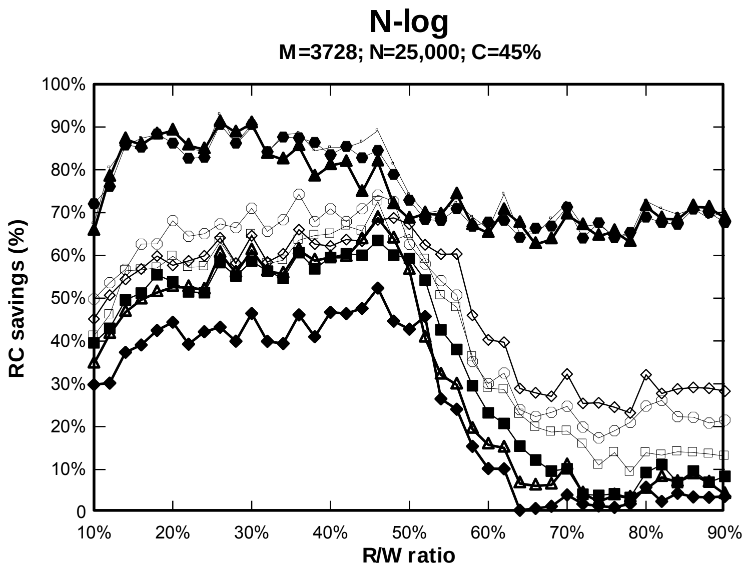

Figure 4.15: RC versus R/W ratio (N-log).

observable gained the most RC savings 38% followed by Greedy with 31%.

Near identical performances were recorded using the N-log (Figure 4.11). 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.

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 4 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 Figures 4.12-4.13) 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.

4.7.3 Impact of Change in 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

┬ (Mean+1.5*Std Dev) ┴ (Mean-1.5*Std Dev) ▒ (Mean+Std Dev – Mean-Std Dev) ■ (Mean) × (Outliers) ✳ (Extremes) ― (Grand mean)

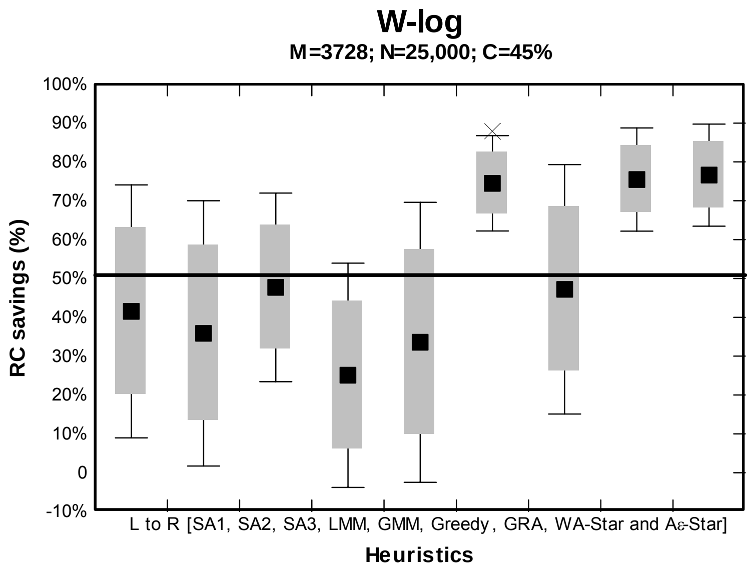

Figure 4.16: RC versus relative performance of heuristics (R/W ratio; W-log).

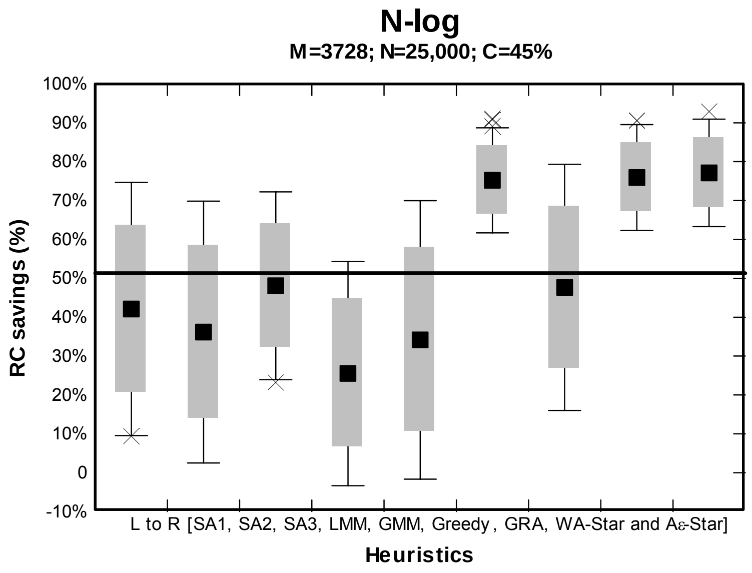

Figure 4.17: RC versus relative performance of heuristics (R/W ratio; N-log).

inadequate. The read and update parameters indeed help in drawing a line between good and marginal algorithms.

Figure 4.14 and Figure 4.15 show the performance of the algorithms using

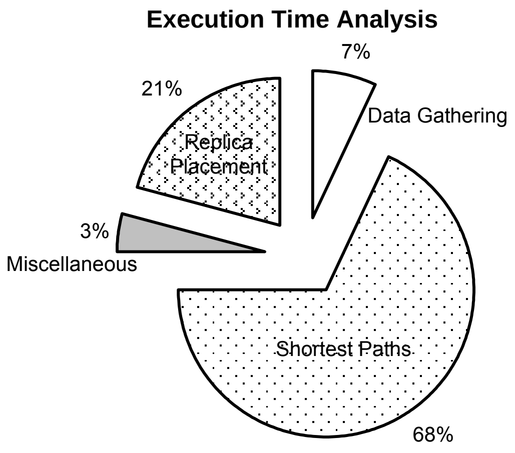

Figure 4.18: Execution time components.

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 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 suboptimal A-star based heuristics suffer from the bound set for their search tree. Thus, by

Table 4.1: Running time in seconds [*C* = 20%, R/W = 0.55] (small 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

Table 4.2: Running time in seconds [*C* = 45%, R/W = 0.85] (medium problem instances).

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

Table 4.3: Running time in seconds [*C* = 75%, R/W = 0.65] (large problem instances).

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

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 Figures 4.16-4.17. 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.

4.7.4 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 Figure 4.18. 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 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 that 21% of the time depicted in Tables 4.1-4.3.

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 seconds (21% of the algorithm termination time). An

┬ (Mean+1.5*Std Dev) ┴ (Mean-1.5*Std Dev) ▒ (Mean+Std Dev – Mean-Std Dev) ■ (Mean) × (Outliers) ✳ (Extremes) ▬ (Grand mean)

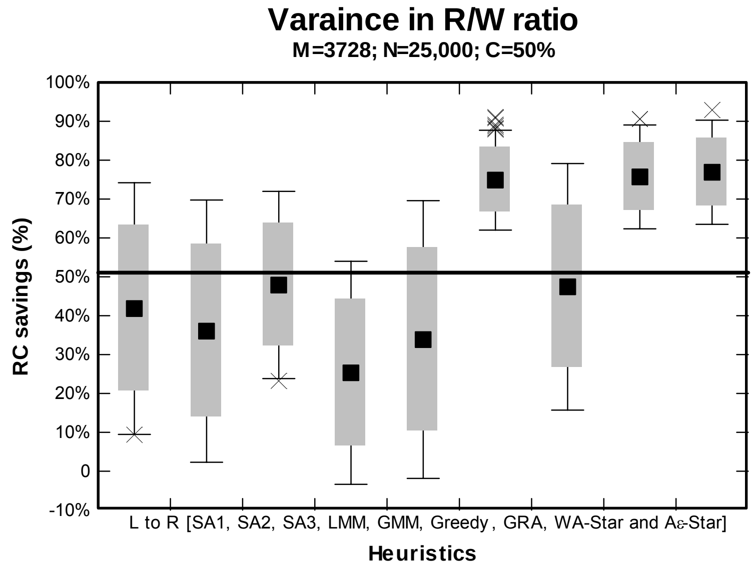

Figure 4.19: RC versus variance in R/W ratio.

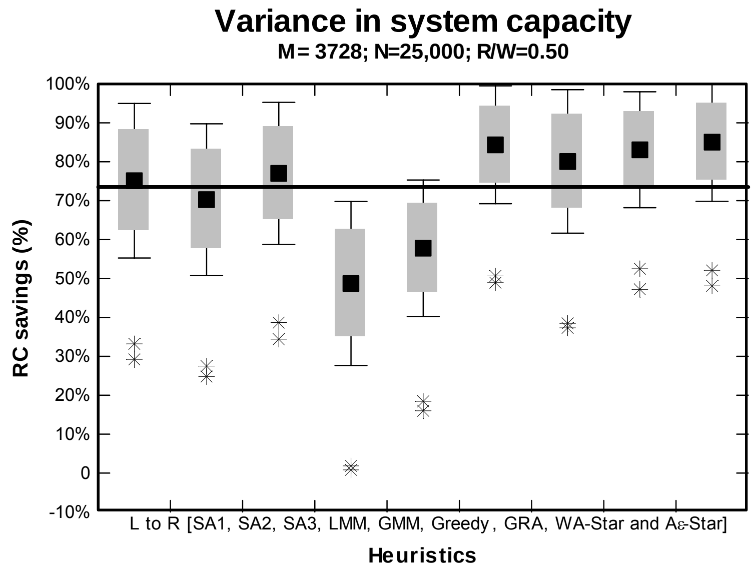

Figure 4.20: RC versus variance in system capacity.

interesting result is also observable in the cases of SA1 and SA2. With soft 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 over head of discarding nodes from the OPEN list.

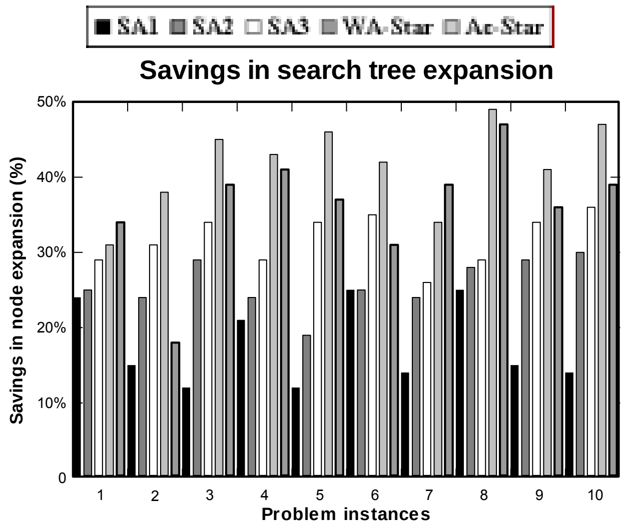

Figure 4.21: Search tree node expansion savings of A-Star based heuristics.

Table 4.4: 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

4.7.5 Summary of Performance

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.

4.7.6 Supplementary Performance Evaluation

Here, we present some supplementary results that strengthen our comparative analysis reported in earlier. The relative performance of the heuristics with variance in R/W ratio and system storage capacity are shown in Figure 4.19 and Figure 4.20, 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 fixed 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 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 (×) represents the outliers, and the asterisks (✳)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 (Figure 4.19), 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

Table 4.5: 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 4.6: 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 4.7: 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-high	Medium	Medium	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	Fast/dynamic with low-medium quality.
Greedy	Low	Low	High	Fast/dynamic with very high quality.
GRA	Medium	Medium	Medium-high	Static with high quality.
WA-Star	Medium	Low-medium	High	Static/dynamic with very high quality.
Aε-Star	Medium	Low-medium	High	Medium fast/dynamic with very high quality.

performed the worst. Figure 4.20 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.

Lastly, we compare the pruning strength of the A-Star based heuristics. Ten problem instances (Table 4.4) were put to test. Figure 4.21 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.

4.7.7 Recap of Evaluation

Table 4.5 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 is indicated in bold. Entries marked with "X" represent that the algorithm could not 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 through investigation of the heuristics under a unifying problem domain is performed. For this purpose we selected ten 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 at 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 [100]. 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 4.6 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 study are summarized in Table 4.7.

4.8 Concluding Remarks

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 study was to study and compare the above mentioned 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 setup that closely mimicked the Internet in its infrastructure and user access patterns. GT-ITM and Inet Internet topology generators were employed 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 object and sites. Based on the experimental data, we were able to comparatively examine the behavior of each of the techniques. For the cases studied in this paper, the relatively simple Greedy heuristic performed extremely well in comparison to other, more complex techniques. 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, R/W 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. Each algorithm has its strengths and weaknesses. 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 study 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.
2. Gather all the system parameters such as, system capacity, number of objects and sites.
3. Observe the access patterns of the clients and extract the spatially and temporally distributed workload so that R/W ratios in corresponding to the system parameters can be obtained.
4. Based on the system parameters and the access patterns determine the relocation time for the replica placement algorithm. If frequent relocations are required, then the portfolio should incorporate algorithms that generate replica schemas in fast turn around time, such as GMM and LMM. If relocation is required only once in a 24-

hour, then the Greedy algorithm would be the best choice. However, if relocation is required, say once every week, then perhaps Aε-Star or WA-Star can be deployed.

CHAPTER 5

ON DESIGNING GAME THEORETICAL REPLICA PLACEMENT TECHNIQUES

Data replication is an essential technique employed to reduce the user perceived access time in distributed computing systems. One can find numerous algorithms that address the data replication or the replica placement problem, each contributing in its own way. These range from the traditional mathematical optimization techniques, such as, linear programming, dynamic programming, etc. to the biologically inspired meta-heuristics. We aim to introduce game theory as a new oracle to tackle the data replication problem. The beauty of the game theory lies in its flexibility and distributed architecture, which is well-suited to address the replica placement problem. We will specifically use action theory (a special branch of game theory) to identify techniques that will effectively and efficiently solve the replica placement problem. Game theory and its necessary properties are briefly introduced, followed by a through and detailed mapping of the possible game theoretical techniques and replica placement problem.

There are two popular models [100] to tackle the replica placement problem 1) centralized replication model and 2) distributed replication model. In the first model, a central body is used to make the decisions about when, where and what to replicate. In the second model, geographically distributed entities (servers, program modules, etc.) are used to make the decisions. Both the techniques have several pros and cons, for instance, the centralized model is a potential point of failure and overloaded by all the computations involved in resolving to a decision. On the other hand, the distributed model suffers from the possibility of mediocre optimization due to the localized view of the distributed entities [58]. A natural way to counteract both the extremities is to view the decision making process as a "semi-distributed" [3] procedure, where all the data intensive computing is performed at the geographically distributed entities, while the final decision on replication is taken by a single entity. This would lessen the burden on the decision making entity, make it more fault-tolerant since in case of a failure it can easily be replaced. It would also improve the overall solution quality since the distributed entities would leverage upon the central body's ability to provide a global snapshot of the system [59].

Game theory has the natural ability to absorb a distributed optimization scenario into its realm [96]. Within the context of the replica placement problem, the geographically distributed entities would be termed as the players, and the central decision making body as the referee, where the players compete to replicate data objects onto their servers so that the users accessing their serves would experience reduced access time. A closer look at this process (competing for data objects and refereeing) reveals a close resemblance between the replica placement problem and auctions. When an object is brought for auction, the bidders in a distributed fashion propose a bid for that object, without knowing what the other bidders are bidding, and the object is allocated to the bidder only when the auctioneer approves it. Of course there is more detail to this process which relies explicitly on the environment, situation, players involved, objects that are up for auction, purpose of the auction. The theory which deals with these details is called auction theory, which is a special branch of game theory [95].

Using game theoretical techniques we can tailor make an auction procedure for a given scenario (problem at hand) and guarantee certain performance criteria, for instance, we can make sure that the players always project the correct worth of an object. This is a difficult problem when the players have to rely on local data, but with the help of game theory, this can be achieved without an extra overhead [61], [94]. To put things into perspective, we will describe how game theory can be used to create techniques for the replica placement problem in distributed computing systems.

5.1 Some Essential Background Material

5.1.1 Background Material on Game Theory

Game theory is widely thought to have originated in the early twentieth century, when von Neumann gave a concrete proof of the min-max theorem [116]. Although, it was the first formally stated major work in this field, the roots of game theory can be traced back to the ancient Babylonian Talmud. The Talmud is a compilation of the ancient laws set forth in the first five centuries A.D. Its traces can be found in various religions and the modern civil and criminal laws. One related problem discussed in the Talmud is the marriage contract problem: A man has three wives. Their marriage contracts specify that in the case of the husband's death, the wives receive 1:2:3 of his property. The Talmud in all mystery gives self-contradictory recommendations. It states that: If the man dies leaving an estate of only 100, there should be an equal division. If the estate is worth 300 it recommends proportional division (50,100,150), while for an estate of 200, it recommends (50,75,75). In 1985, Aumann and Maschler [7] reported that the marriage contract problem and its weird solution discussed in the Talmud are only justifiable via cooperative game theoretical analysis. The foundation of the famous min-max theorem is credited to Waldegrave, who on November 13, 1713 wrote a letter to de Montmort describing a card game *le Her* and his solution [71]. It would take two centuries for Waldegrave's result to be formally acknowledged [116].

Some of the most pioneering results were reported within a year, when Nobel Laureate John Nash made seminal contributions to both cooperative and non-cooperative games. In [91] and [92], Nash proved the existence of a strategic equilibrium for non-cooperative games (Nash Equilibrium). He also proposed that cooperative games were reducible to non-cooperative games. In the next two papers [93], [94] he eventually accomplished that and founded the axiomatic bargaining theory and proved the existence of the Nash Bargaining Solution for cooperative games (a notion similar to the Nash Equilibrium). The beauty of game theory is in its abstractly defined mathematics and notions of optimality. In no other branch of sciences, do we find so many understandable definitions and levels of optimality [96].

Auction theory is a special branch of game theory that deals with biddings and auctions, which have long been an important part of the market mechanisms. Auctions allow buyers to opt for prices often lesser than the original market prices, but they have to compete and in doing so they have to realize their needs and constraints. Analysis of such games began with the pioneering work of Vickery [114]. An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants [66]. For instance, we can formulate an auction as [27]:

1. Bidders send bids to indicate their willingness to exchange goods.
2. The auction may post price quotes to provide summarized information about the status of the price-determination process. (Steps 1 and 2 may be iterated.)
3. The auction determines an allocation and notifies the bidders as to who purchases what from whom at what price. (The above sequence may be performed once or repeated any number of times.)

There are four standard types of auctions [85]:

1. The English auction (also called the oral, open, or ascending-bid auction).
2. The Dutch auction (or descending-bid auction).
3. The first-price sealed-bid auction.
4. The second-price sealed-bid (or Vickrey) auction.

It is to be noted that the English and the Dutch auctions are collectively called progressive auctions; similarly the first-price sealed-bid and second-price sealed-bid auctions are collectively know as sealed-bid auctions.

The English auction is the auction form most commonly used for the selling of goods [29]. In the English auction the price is successfully raised until only one bidder remains. This can be done by having an auctioneer announce prices, or by having bidders call the bids themselves, or by having bids submitted electronically with the current best bid posted [27]. The essential feature of the English auction is that, at any point in time, each bidder knows the level of the current best bid. Antiques and artwork, for example, are often sold by English auction.

The Dutch auction is the converse of the English auction [29]. The auctioneer calls an initial high price and then lowers the price until one bidder accepts the current price. The Dutch auction is used, for instance, for selling cut flowers in Netherlands, fish in Israel and tobacco in Canada.

With the first-price sealed-bid auction, potential buyers submit sealed bids and the highest bidders are awarded items for the price they bid [85]. The basic difference between the first-price sealed-bid auction and the English auction is that, with the English auction, bidders are able to observe their rival's bids and accordingly, if they choose, revise their own bids; with the sealed-bid auction, each bidder can submit only one bid. First-price sealed-bid auctions are used in the auctioning of mineral rights to U.S. government-owned land; they are also sometimes used in the sales of artwork and real estate [66]. Of greater quantitative significance is the use, already noted, of sealed-bid tendering for government procurement contracts [114].

Under the second-price sealed-bid auction, bidders submit sealed bids having been told that the highest bidder wins the item but pays a price equal not to his own bid but to the second-highest bid. While this auction has useful theoretical properties, it is seldom used in practice. The most significant application of this type of auction is found in the selling of FCC bandwidths [66].

5.1.2 Game Theoretical Auction Theory, Views, and Extensions

In contrast with competitive equilibrium theory [48], [118], where players respond solely to summary signals, for instance, prices for different outcomes, in auction theory players act in a game theoretic way, thereby modeling effect their actions will have no other player's actions. This more detailed modeling facilitates the design of predictable systems of interacting players. Specifically, auction theory deals with how to design systems so that certain system-wide criteria, for instance, efficiency, fairness, and stability, emerge in a game theoretic equilibrium. The most widely exploited use of auction theory is in its capability to deal with players that behave in a self-interested (researchers in economic theory often term it as selfish) manner [104], [105]. That is, when players act in their own interest (local optimization) in contrast to the system-wide endeavor (global optimization) [27].

It is also assumed that the players are aware of the protocols of interaction and abide by them. These assumptions are problematic since in a distributed computing environment [27]:

1. Players (by this we mean the software code that models the players) do not have an unbounded computational power that might be required to compute their preferences for all possible equilibria [27].
2. Communication is not necessarily free and can also be prone to errors.
3. Protocols in an open system vary from machine to machine and thus it is practically impossible to equip the players to attain knowledge of every such protocol.

With this said, game theoretical auctions are probably one of the fewest oracles in computer science and economic theory that are flexible and scalable to alterations of extreme nature [84]. For a designer to ensure an equilibrium implemented auction with a certain social choice function, he/she has to predict the strategies players will select. In game theory there are several ways one can achieve that, for instance, dominate strategy, ex post Nash, Bayesian-Nash, ex interim Nash, etc [27]. Of all of them, the most appealing, natural, and straight-forward is the dominate strategy concept [66]. Each player in a dominant strategy has a best response strategy no matter what strategy the other players select [27]. A dominate strategy equilibrium provides a robust solution concept because a player does not need to form beliefs about either other player's rationality or the distribution over the other player types [27]. However, this is not the case when we consider [96]: ex post Nash equilibrium, which requires common knowledge about the player's rationality, Bayesian-Nash equilibrium, which requires the knowledge about the distribution over player types, etc. To bring things into perspective [27]:

1. An example of a dominant strategy implementation is the second-price auction, where the winner always pays the second highest bid's price.
2. An example of an ex post Nash implementation is the first-price open-bid auction, where all the players know exactly which player bids what.
3. An example of the Bayesian-Nash implementation is the first-price sealed-bid auction, where the winner always pays its bid.

As stated previously, the most widely exploited use of auction theory is in its capability to deal with players that behave in a selfish manner. That is, if we are given predictive tools (such as dominant strategy, ex post Nash, ex interim Nash, etc.) and a social choice function, what possible properties can be expected out of the auction designed, given that the players are assumed to be selfish. Recall that we are yet to detail the set of desirable properties for any given auction mechanism. However, we did briefly talk about the utility maximization property for the social choice function. We will soon detail all of them in the subsequent text. For the time being assume we are only concentrating on the utility maximization property [27].

The "direct revelation principle" [86] is the central concept in obtaining results about whether or not a certain property can be expected from a social choice function. The direct revelation principle is a simple reduction technique. For instance, the direct revelation equivalent of the English auction is an action in which the bidding structure follows the ex post Nash equilibrium [27], [84]. That is, the direct revelation implementation asks the players to reveal their valuations and then simulates the English auction with these ex post Nash strategies on the basis of the revealed valuations [27]. The effect is to sell the object to the player with the highest bid for the second highest bid. The beauty of this principle is that any auction mechanism can be transformed into an incentive compatible, direct revelation auction mechanism [62]. By direct we mean that the players' strategy space is restricted to reporting their types and by incentive compatible we mean that the equilibrium strategy for players is truth telling. This principle is important since it allows a focused view on incentive compatible, direct revelation auction mechanisms that can guarantee a certain property. One very famous result using this very principle is the Vickrey-Clarke-Groves (VCG)

[24], [40], [114] auction mechanism that guarantees the utility maximization property [27].

To briefly describe the VCG auction mechanism [27], consider the partitioning of the outcome space into a choice δ and payments p. Outcome $o = (\delta,p)$ defines a choice $\delta \in \Delta$ in the space of feasible choices Δ and payments $p = (p^1, p^2, \ldots, p^M)$ by players. For instance, the choice set can describe all the feasible object allocations to the players, based on the utility function $u^i(\delta,p^i,t^i) = v^i(\delta,t^i) - p^i$, where $v^i(\delta,t^i)$ denotes the value of allocation δ to player i given its type t^i. (It is important to know that the utility functions are always of the form of quasilinear.) The VCG auction mechanism receives claims t^i* from players about their valuations and implements the choice $\delta*$ that maximizes $\sum_i v^i(\delta,t^i*)$. Each player makes payment equal to the second highest bid, i.e., $v^i(\delta,t^i)* - (V(M) - V(M \setminus i))$, where $V(M)$ is the total reported value of $\delta*$ and $V(M\setminus i)$ is the total reported value of $\delta*$ that would be implemented without the player i. Note that the first two terms of the payment align a player's incentives with that of the auction mechanism and make truth revelation a dominant strategy [27]. In equilibrium every player receives as utility the marginal value that it contributes to the system [58].

5.2 Casting Replica Placement Problem into an Incentive Compatible Game Theoretical Auction

To begin we first describe when in the lifespan of the distributed computing system a replication algorithm (an incentive compatible auction if we want to use the correct term) is to be invoked. The answer to that question depends specifically on the system at hand, but usually the replication algorithms are invoked when the system experiences the least amount of queries to access the data objects. This is to ensure that the least amount of users would be affected from the movement of data objects in the system; furthermore it reduces the workload on the entities to compute their preferences towards the objects they prefer to host – remember they are already busy answering all the queries directed to them.

In the subsequent text we will first extract the necessary ingredients from the discussion on auction theory and use them to cast the replica placement problem into an incentive compatible game theoretical auction.

5.2.1 The Ingredients

5.2.1.1 The Basics

The auction mechanism contains M players. Each player i has some private information $t^i \in \Re$. This data is termed as the player's *type*. Only player i has knowledge of t^i. Everything else in the auction mechanism is public knowledge. Let t denote the vector of all the true types $t = (t^1, t^2, \ldots, t^M)$.

5.2.1.2 Communications

Since the players are self-interested (selfish) in nature, they do not communicate the value t^i. The only information that is relayed is the corresponding bid b^i. Let b denote the vector of all the bids ($b = (b^1, b^2, \ldots, b^M)$), and let b^{-i} denote the vector of bids not including player i, *i.e.*, $b^{-i} = (b^1, b^2, \ldots, b^{i-1}, b^{i+1}, \ldots, b^M)$. It is to be understood that we can also write $b = (b^{-i}, b^i)$.

5.2.1.3 Components

The auction mechanism has two components 1) the algorithmic output $o(\cdot)$, and 2) the payment mapping function $p(\cdot)$.

5.2.1.4 Algorithmic Output

The auction mechanism allows a set of outputs O, based on the output function which takes in as the argument, the bidding vector, *i.e.*, $o(b) = \{o^1(b), o^2(b), \ldots, \mathrm{o}^M(b)\}$, where $o(b) \in O$. This output function relays a unique output given a vector b. That is, when $o(\cdot)$ receives b, it generates an output which is of the form of allocations $o^i(b)$. Intuitively it would mean that the algorithm takes in the vector bid b and then relays to each player its allocation.

5.2.1.5 Monetary Cost

Each player i incurs some monetary cost $c^i(t^i,o)$, *i.e.*, the cost to accommodate the allocation $o^i(b)$. This cost is dependent upon output and player's private information.

5.2.1.6 Payments

To offset c^i, the auction mechanism makes a payment $p^i(b)$ to player i. A player i always attempts to maximize its profit (utility) $u^i(t^i,b) = p^i(b) - c^i(t^i,o)$. Each player i

cares about the other players' bid only insofar as they influence the outcome and the payment. While t^i is only known to player i, the function c^i is public. (Note that when we were previously describing the properties of auctions, the payments were made by the players and not the auction mechanism. That is fine in that context since the incentive for the players there was to acquire the object. In the context of the DRP, since the players have to conform to the global optimization criteria and host the objects, an incentive for the players would be to receive payments for hosting objects rather than making payments.)

5.2.1.7 Bids

Each player i is interested in reporting a bid b^i such that it maximizes its profit, regardless of what the other players bid (dominant strategy), *i.e.*, $u^i(t^i,(b^{-i},t^i)) \geq u^i(t^i,(b^{-i},b^i))$ for all b^{-i} and b^i.

5.2.1.8 The incentive Compatible Auction Mechanism

We now put all the pieces together. An incentive compatible auction mechanism consists of a pair $(o(b),p(b))$, where $o(\cdot)$ is the output function and $p(\cdot)$ is the payment mapping function. The objective of the auction mechanism is to select an output o, that optimizes a given objective function $f(\cdot)$.

5.2.1.9 Desirable Properties

In essence, we desire our auction to exhibit the utility maximization property. In economic theory a utility maximization property is also known as the efficient outcome of the auction mechanism [26]. In the subsequent text we will consider other properties, but for the time being let us limit our discussion to the utility maximization property. Remember that a utility maximization property is only attainable when an auction is an incentive compatible auction [64].

5.2.2 The Casting

We follow the same pattern as discussed in Section 5.2.1.

5.2.2.1 The Basics

The distributed system described in Chapter 3 is considered, where each server is represented by a player, *i.e.*, the auction mechanism contains M players. In the context of the replica placement problem, a player holds two key elements of information 1) the available server capacity ac^i and 2) the access frequencies (both read r_k^i and write w_k^i). Let us consider what possible cases for information holding there are:

1. Replica placement problem [π]: Each player i holds the access frequencies $\{r_k^i, w_k^i\} = t^i$ associated with each object k as private information, where as the available server capacity ac^i and everything else (this includes all the auction related functions, network and system parameters) is public knowledge.
2. Replica placement problem [σ]: Each player i holds the available server capacity $ac^i = t^i$ as private information, where as the access frequencies $\{r_k^i, w_k^i\}$ and everything else is public knowledge.
3. Replica placement problem [π,σ]: Each player i holds both the access frequencies

$\{r_k^i, w_k^i\}$ and the server capacity ac^i as private information $\{ac^i, \{r_k^i, w_k^i\}\} = t^i$, where as everything else is public knowledge.

Intuitively, if players know the available server capacities of other players, that gives them no advantage whatsoever. However, if they come about to know their access frequencies, then they can modify their valuations and alter the algorithmic output. Everything else such as the network topology, latency on communication lines, and even the server capacities can be public knowledge. Therefore, replica placement problem [π] is the only natural choice.

5.2.2.2 Communications

The players in the auction mechanism are assumed to be selfish and therefore, they project a bid b^i to the auction mechanism.

5.2.2.3 Components

The auction mechanism has two components 1) the algorithmic output $o(\cdot)$, and 2) the payment mapping function $p(\cdot)$.

5.2.2.4 Algorithmic Output

In the context of the DRP, the replication algorithm accepts bids from all the players, and outputs the maximum beneficial bid, *i.e.*, the bid that incurs the minimum cost to place replicas (Equation 3.3). We will give a detailed description of the algorithm in the later text.

5.2.2.5 Monetary Cost

When an object is allocated (for replication) to a player i, the player becomes responsible to entertain (read and write) requests to that object. For example, assume object k is replicated by player i. Then the amount of traffic that the player's server has to entertain due to the replication of object k is exactly equivalent to the replication cost, i.e., $c^i = R_k^i + W_k^i$. This fact is easily deducible from Equation 3.4.

5.2.2.6 Payments

To offset c^i, the auction mechanism makes a payment $p^i(b)$ to player i. This payment is chosen by the auction mechanism such that it eliminates incentives for misreporting by imposing on each player the cost of any distortion it causes. The payment for player i is set so that i's report cannot effect the total payoff to the set of other players (excluding player i), M-i. With this principle in mind, let us derive a formula for the payments. To capture the effect of i's report on the outcome, we introduce a hypothetical *null report*, which corresponds to player i reporting that it is indifferent among the possible decisions and cares only about payments. When player i makes the null report, the auction optimally chooses the allocation $o(t^{-i})$. The resulting total value of the decision for the set of players M-i would be $V(M\text{-}i)$, and the auction might also "collect" a payment $h^i(t^{-i})$ from player i. Thus, if i makes a null report, the total payoff to the players in set M-i is $V(M\text{-}i) - h^i(t^{-i})$.

The auction is constructed so that this, $V(M\text{-}i) - h^i(t^{-i})$, amount is the total payoff to those players regardless of i's report. Thus, suppose that when the reported type is t,

i's payment is $p^i(t) + h^i(t^{-i})$, so that $p^i(t)$ is i's additional payment over what i would pay if it made the null report. The decision $o(t)$ generally depends on i's report, and the total payoff to members of $M\text{-}i$ is then $\sum_{i\in M-i} v^i(o(t),t^i) + p^i(t) + h^i(t^{-i})$. We equate this total value with the corresponding total value when player i makes the null report:

$$\sum_{i\in M-i} v^i\left(o(t),t^i\right)+p^i(t)+h^i\left(t^{-i}\right)=V(M-i)+h^i\left(t^{-i}\right). \tag{5.2}$$

Using Equation 5.2, we solve for the extra payment as:

$$p^i(t)=V(M-i)-\sum_{i\in M-i} v^i\left(o(t),t^i\right). \tag{5.3}$$

$$p^i(t)=\sum_{i\in M-i} v^i\left(o(M-i),t^i\right)-\sum_{i\in M-i} v^i\left(o(t),t^i\right). \tag{5.4}$$

According to Equation 5.4, if player i's report leads to a change in the decision o, then i's extra payment $p^i(t)$ is specified to compensate the members of $M\text{-}i$ for the total losses they suffer on the account [58].

The derived payment procedure is in its most general form. A careful observation would reveal that its special cases include every possible payment procedure. The most famous of them all is the Vickrey payment. To say the least we will show the derived payment procedure is equivalent to Vickrey payments. A player's value for any decision depends only on the objects that the player acquires, and not on the objects acquired by other players. That is, $v^i(t^i) = 1$ if the player acquires the object and $v^i(t^i) = 0$ otherwise. Since the loosing players are not pivotal [64] (because their presence does not affect the allocation o), they obtain zero payments in our mechanism. According to Equation 5.4, the price a winning player pays in the (derived) payment procedure is equal to the difference between the two numbers. The first number is the maximum total value of all the other players, when *i* does not participate, which is $\max_{j \neq i} v^i$. The second number is the total value of all the other players when *i* wins, which is zero. Thus, when *i* wins it pays $\max_{j \neq i} v^i$, which is equal to the second highest valuation. This is exactly the Vickrey payment [110].

5.2.2.7 Bids

Each player *i* reports a bid that is the direct representation of the true data that it holds. Therefore, a bid b^i is equivalent to $1/\{R_k^i + W_k^i\}$. That is, the lower the replication cost the higher is the bid and the higher are the chances for the bid b^i to win.

In essence, the incentive compatible auction mechanism $(o(b),p(b))$, takes in the vector of bids *b* from all the players, and selects the highest bid. The highest bidder is allocated the object *k* which is added to its allocation set o^i. The auction mechanism then pays the bidder p^i. This payment is equivalent to the Vickrey payments and compensates the cost incurred (due to the entertainment of access requests for object *k* by users) by the player to host the object at its server. A pseudo-code for an incentive compatible auction mechanism is given in Figure 5.1.

5.2.2.8 Description of Psuedo-code

We maintain a list L^i at each server. This list contains all the objects that can be replicated by player *i* onto server S^i. We can obtain this list by examining the two constraints of the DRP. List L^i would contain all the objects that have their size less then the total available space b^i. Moreover, if server S^i is the primary host of some object *k*’,

An incentive compatible auction mechanism

Initialize:
LS, L^i, T_k^i, M, MT

```
01 WHILE $LS \neq$ NULL DO
02   $OMAX$ = NULL; $MT$ = NULL; $P^i$ = NULL;
03       PARFOR each $S^i \in$ LS DO
04               FOR each $O_k \in L^i$ DO
05                       $T_k^i$ = compute ($B_k^i$);  /*Compute the valuation */
06               ENDFOR
07           $t^i$ = argmax$_k$($T_k^i$);
08           SEND $t^i$ to M; RECEIVE at $M$ $t^i$ in $MT$;
09       ENDPARFOR
10 OMAX = argmax$_k$($MT$);    /*Choose the global dominate valuation*/
11 DELETE $k$ from $MT$;
12 $P^i$ = argmax$_k$($MT$);             /*Calculate the Vickrey payment*/
13 BROADCAST $OMAX$;
14 SEND $P^i$ to $S^i$; RECEIVE at $S^i$     /*Ask the winning agent to pay this amount*/
15 SEND $P^i$ to $M$; RECEIVE at M      /*Send the required payment*/
16 Replicate $O_{OMAX}$;
17 $b^i$=$b^i$ - $o_k$;                    /*Update capacity*/
18 $L^i$ = $L^i$ - $O_k$;                  /*Update the list*/
19 IF $L^i$ = NULL THEN SEND info to $M$ to update $LS$ = $LS$ - $S^i$;      /*Update mechanism players*/
20       PARFOR each $S^i \in$ LS DO
21           Update $NN^i_{OMAX}$          /*Update the nearest neighbor list*/
22       ENDPARFOR                /*Get ready for the next round*/
23 ENDWHILE
```

Figure 5.1: Pseudo-code for an incentive compatible auction mechanism.

then *k*' should not be in L^i. We also maintain a list *LS* containing all servers that can replicate an object, *i.e.*, $S^i \in LS$ if $L^i \neq$ NULL. The algorithm works iteratively. In each step the auction mechanism asks all the players to send their preferences (first **PARFOR** loop). Each player *i* recursively calculates the true data of every object in list L^i. Each player then reports the dominant true data (line 08). The auction mechanism receives all the corresponding entries, and then chooses the best dominant true data. This is broadcasted to all the players, so that they can update their nearest neighbor table NN_k^i, which is shown in Line 21 (NN^i_{OMAX}). The object is replicated and payments made to the player. The auction progresses forward till there are no more players interested in acquiring any data for replication.

5.2.3 Further discussion on the Casting Process

The mechanism described in Section 5.2 illustrates the usage of the auction theory as a possible solution towards the replica placement problem with the property of dominating strategy. This same process with minor modifications can be used to guarantee other auction properties applied to the replica placement problem. We give a brief description of some of the properties in the subsequent text, but for details, the readers are encouraged to see some of the work performed by the authors that explicitly detail these properties, and the subsequent Chapters.

5.2.3.1 Pareto Optimality

Implementing an outcome that is not pareto dominated by any other outcomes, so no other outcomes make one player better off while making other players worst [27]. Details on a pareto optimal auction applied to the replica placement problem can be found in Chapter 6.

5.2.3.2 Maximum Utility to a Particular Player

Maximizing the expected utility to a single player, typically the central decision making body, across all possible scenarios. This type of setting is very useful when considering revenue maximization scenarios. Details on a maximum utility to a particular player auction applied to the replica placement problem can be found in Chapter 7.

5.2.3.3 Deliberate Discrimination of Allocation

Maximize the system utilization by revoking allocations if deemed necessary. This type of property is very useful when considering dynamic scenarios, where it often warrants revoking a decision since the system parameters may change drastically during the computation of a decision [27]. Details on a deliberate discrimination of allocation auction applied to the replica placement problem can be found in Chapter 8.

5.2.3.4 Budget Balance

A budget balanced auction is when the total payments made or received by the players exactly equals zero. This property is important since the money is not injected or removed from the system. If the payments made or received by the players equal to zero, then the auction is termed as a strict budget balance auction [27]. On the other hand, if the payments made or received by the players does not equal to zero but it is non-negative, then the auction is termed as a weak budget balance auction. (In a weak budget balance auction, the auction does not run at a loss.) One can also consider an exante budget balance auction, in which the auction is balanced on average, and an expost budget balance auction, in which the auction is balanced at all times. Details on a budget balance auction applied to the replica placement problem can be found in Chapter 9.

Budget balance is especially important in systems that must be self-sustaining and require no external benefactor to input money or central authority to collect payments [26], [27]. For instance, a distributed system should always be a budget balanced system, since money has no literal meaning in the system – it is there just to drive the optimization process, not the system as a whole.

5.3 Experimental Comparative Analysis

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory. The experimental evaluations were targeted to benchmark the placement policies. The solution quality in all cases, was measured according in terms of the OTC (or RC or NTC) percentage that was saved under the replication scheme found by the technique, compared to the initial one, *i.e.*, when only primary copies exist.

5.3.1 Comparative Techniques

For comparison, we selected three various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The techniques studied include efficient branch and bound based technique (Aε-Star [57]). The algorithms proposed in [58], [75], [78], and [100] are the only ones that address the problem domain similar to ours. We select from [100] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [75]); thus, we indirectly compare with 4 additional approaches as well. From [78] we choose the

Genetic based algorithm (GRA), which exhibits extreme robustness under various changing scenarios. For rebuttal, we briefly detail the comparative techniques as below:

1. Aε-Star: In [57] the authors proposed a 1+ε admissible A-Star based technique called Aε-Star. This technique uses two lists: OPEN and FOCAL. The FOCAL list is the sub-list of OPEN, and only contains those nodes that do not deviate from the lowest *f* node by a factor greater than 1+ε. The technique works similar to A-Star, with the exception that the node selection (lowest *h*) is done not from the OPEN but from the FOCAL list. 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+ε of the A-Star are expanded.
2. Greedy: We modify the greedy approach reported in [100], to fit our problem formulation. The greedy algorithm works in an iterative fashion. In the first iteration, all the *M* servers 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 server that yields the lowest cost of replication for the object *i*. In the second iteration, the location for the second server is considered. Based on the choice of object *i*, the algorithm now would identify the second server for replication, which, in conjunction with the server already picked, yields the lowest replication cost. 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. The readers will

immediately realize that derived incentive compatible auction mechanism works similarly to the Greedy algorithm. This is true; however, the Greedy approach does not guarantee optimality even if the algorithm is run on the very same problem instance. Recall that Greedy relies on making combinations of object assignments and therefore, suffers from the initial choice of object selection (which is done randomly). This is never the case with the derived auction procedure, which identifies optimal allocations in every case.

3. GRA: In [78], the authors proposed a genetic algorithm based heuristic called GRA. GRA provides good solution quality, but suffers from slow termination time. This algorithm was selected since it realistically addressed the fine-grained data replication using the same problem formulation as undertaken in this article.

From here onwards, we will acronym the **i**ncentive **c**ompatible **a**uction **m**echanism derived exclusively for the replica placement problem as I-CAM.

5.3.2 Comparative Analysis

We record the performance of the techniques using the access logs and 80 topologies. Note that each point represents the average performance of an algorithm over 80 topologies and 88 days of the access log. (The details on the infrastructure and the workload have already been discussed previously.) Below we detail our experimental findings.

5.3.2.1 Impact of Change in the Number of Servers and Objects

We study the behavior of the placement techniques when the number of servers increase (Figure 5.2), by setting the number of objects to 25,000, while in Figure 5.3, we study the behavior when the number of objects increase, by setting the number of servers to 3718. For the first experiment we fixed C = 35% and R/W = 0.25. We intentionally chose a high workload so as to see if the techniques studied successfully handled the extreme cases. By adding a server in the network, we introduce additional traffic due to its local requests, together with more storage capacity to be used for replication. I-CAM balances and explores these diverse effects, so as to achieve highest OTC savings. GRA showed the worst performance along all the techniques. It showed an initial gain, since with the increase in the number of servers the population permutations increase exponentially, but with the further increase in the number of servers this phenomenon is not so observable as all the essential objects are already replicated. The top performing techniques (I-CAM, Greedy and Aε-Star) showed an almost constant performance increase (after the initial surge in OTC savings). GRA also showed a similar trend but maintained lower OTC savings. This was in line with the claims presented in [57] and [78].

To observe the effect of increase in the number of objects in the system, we chose a softer workload with C = 65% and R/W = 0.70. The intention was to observe the trends for all the techniques under various workloads. The increase in the number of objects has diverse effects on the system as new read/write patterns (since the users are

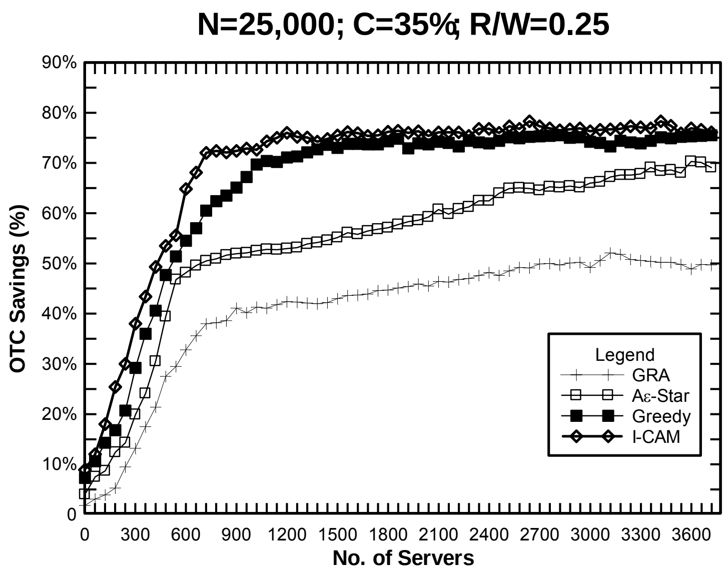

Figure 5.2: OTC savings versus number of servers.

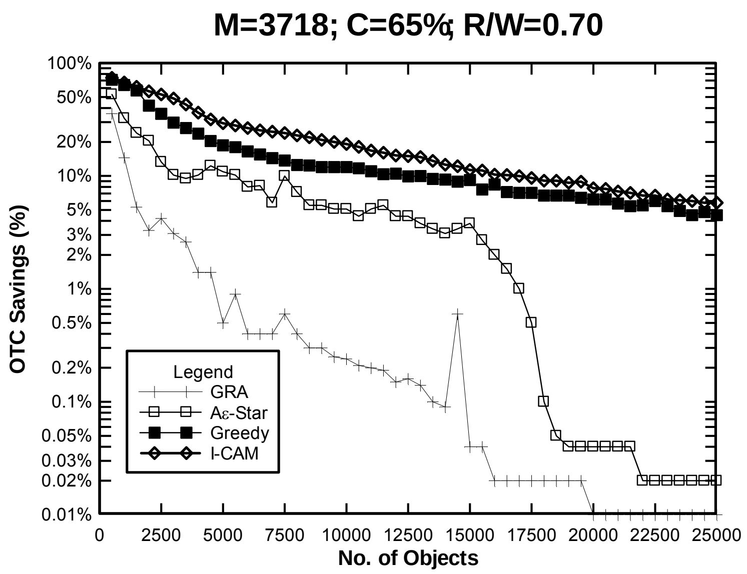

Figure 5.3: OTC savings versus number of objects

offered more choices) emerge, and also the strain on the overall storage capacity of the system increases (due to the increase in the number of replicas). An effective replica allocation method should incorporate both the opposing trends. From the plot, the most surprising result came from GRA. It dropped its savings from 47% to 0.01%. This was contradictory to what was reported in [78]. But there the authors had used a uniformly

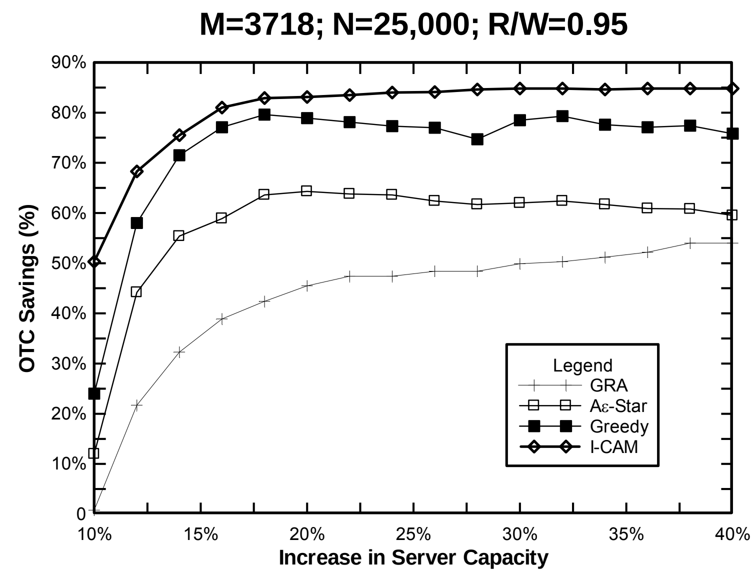

Figure 5.4: OTC savings versus capacity.

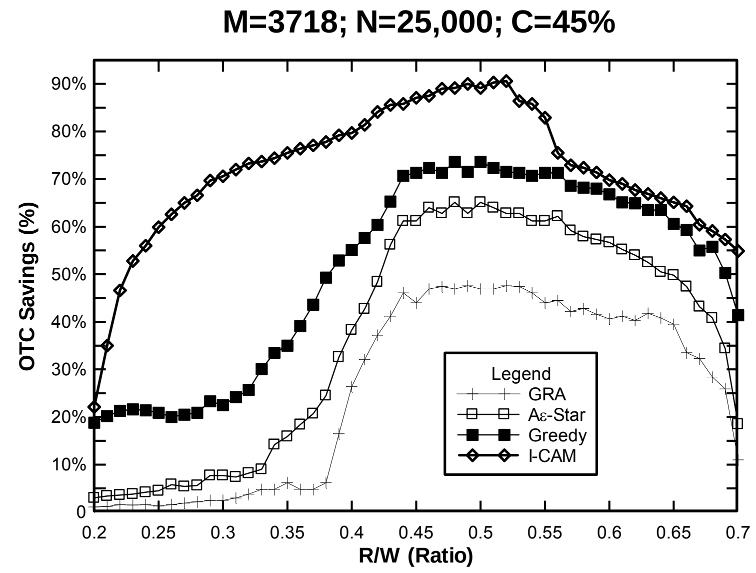

Figure 5.5: OTC savings versus read/write ratio.

distributed link cost topology, and their traffic was based on the Zipf distribution [123]. While the traffic access logs of the Soccer World Cup 1998 are more or less double-Pareto in nature [6]. In either case the exploits and limitations of the technique under discussion are obvious. The plot also shows a near identical performance by Aε-Star and Greedy. The relative difference among the two techniques was less than 7%. However, Greedy did maintain its dominance. From the plots the supremacy of I-CAM is observable. (Figure 5.3 is deliberately shown with a log (OTC savings) scale to better appreciate the performances of the techniques.)

5.3.2.2 Impact of Change in the system Capacity

Next, we observe the effects of increase in storage 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. This is observable in Figure 5.4, which shows the performance of the algorithms. GRA once again performed the worst. The gap between all other approaches was reduced to within 15% of each other. I-CAM and Greedy showed an immediate initial increase (the point after which further replicating objects is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most OTC savings (49%) followed by Greedy with 44%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in capacity from 10% to 18%, resulted in 3.75 times (on average) more replicas for all the algorithms.

5.3.2.3 Impact of Change in the Read/Write Frequencies

Next, we observe the effects of increase in the read and write frequencies. Since these two parameters are complementary to each other, we describe them together. To observe the system utilization with varying read/write frequencies, we kept the number of servers and objects 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 in the system requires the replicas be placed as close as to the primary server 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. The plot in Figure 5.5 shows the results of read/write ratio against the OTC savings. A clear classification can be made between the algorithms. I-CAM and Greedy incorporate the increase in the number of reads by replicating more objects and thus savings increased up to 88%, while GRA gained the least of the OTC savings of up to 42%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial selection of gene population (for details see [78]). Moreover, GRA maintains a localized network perception. Increase in updates result in objects having

Table 5.1: Running time of the replica placement methods in seconds [*C*=45%, *R/W*=0.85].

Problem Size	Greedy	GRA	Aε-Star	I-CAM
M=2500, *N*=15,000	310.14	491.00	399.63	**185.22**
M=2500, *N*=20,000	330.75	563.25	442.66	**201.75**
M=2500, *N*=25,000	357.74	570.02	465.52	**240.13**
M=3000, *N*=15,000	452.22	671.68	494.60	**284.34**
M=3000, *N*=20,000	467.65	726.75	498.66	**282.35**
M=3000, *N*=25,000	469.86	791.26	537.56	**303.32**
M=3718, *N*=15,000	613.27	883.71	753.87	**332.48**
M=3718, *N*=20,000	630.39	904.20	774.31	**390.90**
M=3718, *N*=25,000	646.98	932.38	882.43	**402.23**

Table 5.2: Average OTC (%) savings under some randomly chosen problems.

Problem Size	Greedy	GRA	Aε-Star	I-CAM
M=100, *N*=1000 [*C*=20%,*R/W*=0.75]	71.46	85.77	86.28	**89.45**
M=200, *N*=2000 [*C*=20%, *R/W*=0.80]	84.29	78.30	79.02	**84.76**
M=500, *N*=3000 [*C*=25%, *R/W*=0.95]	68.50	70.97	67.53	**71.43**
M=1000, *N*=5000 [*C*=35%, *R/W*=0.95]	88.09	67.56	78.24	**88.30**
M=1500, *N*=10,000 [*C*=25%, *R/W*=0.75]	89.34	52.93	76.11	**89.75**
M=2000, *N*=15,000 [*C*=30%, *R/W*=0.65]	67.93	51.02	52.42	**75.32**
M=2500, *N*=15,000 [*C*=25%, *R/W*=0.85]	77.35	71.75	73.59	**81.12**
M=3000, *N*=20,000 [*C*=25%, *R/W*=0.65]	76.22	65.89	73.04	**82.31**
M=3500, *N*=25,000 [*C*=35%, *R/W*=0.50]	66.04	59.04	67.01	**71.21**
M=3718, *N*=25,000 [*C*=10%, *R/W*=0.40]	76.34	63.19	76.02	**79.21**

decreased local significance (unless the vicinity is in close proximity to the primary location). On the other hand, I-CAM, Aε-Star and Greedy never tend to deviate from their global (or social) view of the problem.

5.3.2.4 Running Time

Lastly, we compare the termination time of the algorithms. Various problem instances were recorded with *C* = 45% and *R/W* = 0.85. The entries in Table 5.1 made bold represent the fastest time recorded over the problem instance. It is observable that I-CAM terminated faster than all the other techniques, followed by Greedy, Aε-Star, and GRA.

5.3.2.5 Summary of Experimental Results

Table 5.2 shows the quality of the solution in terms of OTC percentage for 10 problem instances (randomly chosen), each being a combination of various numbers of server and objects, with varying storage capacity and update ratio. For each row, the best result is indicated in bold. The proposed I-CAM steals the show in the context of solution quality, but Greedy and Aε-Star do indeed give a good competition.

In summary, based on the solution quality alone, the replica allocation methods can be classified into four categories: 1) High performance: I-CAM; 2) Medium-High performance: Greedy; 3) Medium performance: Aε-Star; 5) Low performance: GRA. Considering the execution time, I-CAM and Greedy did extremely well, followed by Aε-Star and GRA.

5.4 Concluding Remarks

Replicating data across a distributed computing system can potentially reduce the user perceived access time which in turn reduces latency, adds robustness and increases data availability. Our focus here was to show how game theoretical auctions can be used to identify techniques for the replica placement problem in distributed computing systems. A semi-distributed technique based on a game theoretical auction was proposed for the replica placement problem which had the added property that it maximized the utility of all the players involved in the system – an incentive compatible auction mechanism (I-CAM).

I-CAM is a protocol for automatic replication and migration of objects in response to demand changes. It aims to place objects in the proximity of a majority of requests while ensuring that no servers become overloaded. The infrastructure of I-CAM was designed such that, each server was required to present a list of data objects that if replicated onto that server would bring the communication cost to its minimum. These lists were reviewed at the central decision body which gave the final decision as to what object are to be replicated onto what servers. This semi-distributed infrastructure takes away all the heavy processing from the central decision making body and gives it to the individual servers. For each object, the central body is only required to make a binary decision: (0) not to replicate or (1) to replicate.

To compliment our theoretical results, we compared I-CAM with three conventional replica allocation methods namely: (1) branch and bound, (2) greedy, and (3) genetic. The experimental setups were designed in such a fashion that they resembled real world scenarios. We employed GT-ITM and Inet to gather 80 various Internet topologies based on flat, link distance, power-law and hierarchical transit-stub models, and used the traffic logs collected at the Soccer World Cup 1998 website for mimicking user access requests. The experimental study revealed that the proposed I-CAM technique improved the performance relative to other conventional methods in four ways.

1. The number of replicas in a system was controlled to reflect the ratio of read versus write access. To maintain concurrency control, when an object is updated, all of its

replicas need to be updated simultaneously. If the write access rate is high, there should be few replicas to reduce the update overhead. If the read access rate is overwhelming, there should be a high number of replicas to satisfy local accesses.

2. Performance was improved by replicating objects to the servers based on locality of reference. This increases the probability that requests can be satisfied either locally or within a desirable amount of time from a neighboring server.
3. Replica allocations were made in a fast algorithmic turn-around time.
4. The complexity of the data replication problem was decreased by multifold. I-CAM limits the complexity by partitioning the complex global problem of replica allocation, into a set of simple independent sub problems. This approach is well suited to the large-scale distributed computing systems that are composed of autonomous agents which do not necessarily cooperate to improve the system wide goals.

All the above improvements were achieved by a simple, semi-distributed, and autonomous I-CAM.

CHAPTER 6

A PARETO OPTIMAL GAME THEORETICAL REPLICA PLACEMENT TECHNIQUE

Here we derive a pareto optimal replica placement technique based on the extended form of Vickrey auction called the N+1st price auction. Specifically, we present an adaptive auction mechanism for replication of objects in a distributed system. The mechanism is adaptive in the sense that it changes the replica schema of the objects by continuously moving the schema towards an optimal one, while ensuring object concurrency control.

6.1 Motivation

As a rule of thumb, a replica placement technique should pursue the following line of action.

1. Determine the Network topology.
2. Specify the objects that are to be replicated.
3. Obtain the access frequencies of the objects. The access frequencies are either known *apriori* or determined using some prediction techniques.
4. Based on the above information, employ an *algorithmic technique* to replicate objects based on some *optimization criteria* and *constraints*.

5. Finally, determine a redirection method that sends client requests to the best *replicator* that can satisfy them.

Based on the above passage, an effective replica placement technique determines the replica allocation which gives the highest data accessibility in the whole network. If the network topology is comprised of M sites which are connected (directly or indirectly) to each other and N denotes the number of data objects that are specified for replication, then, the number of possible combinations of replica allocation is expressed by the following expression:

$$\left\{\frac{N!}{(N-C)!}\right\}^{M},$$

where C is the overall memory capacity of M sites.

In order to determine the optimal allocation among all possible combinations, we must analytically find a combination which gives the highest data accessibility considering the following parameters:

1. Access frequencies from each site to each data object.
2. The probability that each site's memory capacity remains unchanged.
3. The probability that the network connectivity remains unchanged.

Even if some looping is possible the computational complexity is very high, and this calculation must be done every time when either of the above three parameters change. Moreover, among the above three parameters, the later two cannot be formulated in practical because they follow no known phenomenon.

For these reasons, we take the following approach:

1. Replicas are relocated in a specific period (*relocation period*).
2. At every relocation period, replica allocation is determined based on the access frequency from each site to each data object and the network topology at the moment.

Based on this approach we propose a pareto optimal game theoretical technique that effectively and efficiently determines a replica schema that is competitive, scalable and simple compared to other conventional (heuristics) techniques.

6.2 The Mechanism (NPAM)

We term the proposed resource allocation mechanism as NPAM an acronym for N+1st Price Auction Mechanism. In the auction setup each primary copy of an object *k* is a player. A player *k* can perform the necessary computations on its strategy set by using the site (where it resides) P_k's processor. At each given instance a (sub)-auction takes place at a particular site *i* chosen in a round robin fashion from the set of *M* sites. These auctions are performed continuously throughout the system's life, making it a self evolving and self repairing system. However, for simulation purposes ("cold" network [77]) we discrete the continuum solely for the reason to observe the solution quality.

Each player *k* competes through bidding for memory at a site *i*. Many would argue that memory constraints are no longer important due to the reduced costs of memory chips. However, replicated objects (just as cached objects) reside in the memory (primary storage) and not in the media (secondary storage) [100]. Thus, there will always be a need to give priority to objects that have higher access (read and write) demands. Moreover, memory space regardless of being primary or secondary is limited.

Each player *k*'s strategy is to place a replica at a site *i*, so that it maximizes its (the object's) benefit function. The benefit function gives more weight to the objects that incur reduced RC in the system:

$$B_k^i = R_k^i - \left(\sum_{x=1}^{M} w_k^x o_k c\left(i, P_k\right) - W_k^i \right). \tag{6.1}$$

The above value represents the expected benefit (in RC terms), if O_k is replicated at S_i. This benefit is computed using the difference between the read and update cost. Negative values of B_k^i mean that replicating O_k, is inefficient from the "local view" of S_i (although it might reduce the global RC due to bringing the object closer to other servers). The pseudo-code for the N+1st price auction is given in Figure 6.1.

We maintain a list L^i at each server. The list contains all the objects that can be replicated at S_i (*i.e.*, the remaining storage capacity b^i is sufficient and the benefit value is positive). We also maintain a list *LS* containing all servers that can replicate an object. In other words, $S_i \in \text{LS}$ if and only if $L^i \neq \text{NULL}$. The auction mechanism performs in steps. In each step a server S_i is chosen from *LS* in a round-robin fashion. Each player $k \in O$ calculates the benefit function of object. The set *O* represents the collection of players that are legible for participation. A player *k* is legible if and only if the benefit function value obtained for site S_i is the maximum of among all the other benefit function values for sites other than *i*, *i.e.*, $S_i \geq S_{-i}$. This is done in order to

```
N+1st Price Auction Mechanism
Initialize:
01 $LS$, $L^i$.
02 WHILE $LS \neq$ NULL DO
03       SELECT $S^i \in$LS                    /*Round-robin fashion */
04             FOR each $k \in O$ DO
05                   $B_k$ = compute ($B_k^i$);        /*compute the benefit*/
06                   Report $B_k$ to $S_i$ which stores in array $B$;
07             END FOR
08       WHILE $b_i \geq 0$
09         $B_k$ = argmax$_k$($B$);              /*Choose the best offer*/
10         Extract the info from $B_k$ such as $O_k$ and $o_k$;
11         $b_i = b_i - o_k$;                   /*Calculate available space and termination condition*/
12         Payment = $B_k$;                  /* Maintain N+1st price */
13         IF $b_i < 0$ THEN EXIT WHILE ELSE
14         $L^i = L^i - O_k$;                  /*Update the list*/
15         Update $NN^i_{OMAX}$               /*Update the nearest neighbor list*/
16         IF $L^i$ = NULL THEN SEND info to $M$ to update $LS = LS - S^i$;
17         Replicate $O_k$;
18     END WHILE
19     $S_i$ asks all successful bidders to pay $B_k$
20 END WHILE
```

Figure 6.1: Pseudo-code for N+1st Price Auction Mechanism (NPAM).

suppress mediocre bids, which, in turn improves computational complexity. It is to be noted that in each step L^i together with the corresponding nearest server value NN_k^i, are updated accordingly.

Theorem 6.1: *NPAM takes $O(MN^2)$ time.*

Proof: The worst case execution time of the algorithm is when each server has sufficient capacity to store all objects and the update ratios are low enough so that no object incurs negative benefit value. In that case, the while-loop (02) performs M iterations. The time complexity for each iteration is governed by the for-loop in (04) and the while loop in (08) ($O(N^2)$ in total). Hence, we conclude that the worst case running time of the algorithm is $O(MN^2)$. ■

6.3 Experimental Comparative Analysis

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory using the setup described in Chapter 3. The experimental evaluations were targeted to benchmark the placement policies. The solution quality in all cases, was measured according to 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.

For comparisons, we selected three various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. We chose 1) from [57] the efficient branch-and-bound based technique (Aε-Star), 2) from [78] the genetic algorithm based technique (GRA) which showed excellent adaptability against skewed workload, 3) and from [100] the famous greedy approach (Greedy).

Table 6.1 (best times shown in bold) shows the algorithm execution times. The number of sites was kept constant at 500, and the number of objects was varied from 1350 to 2000. With maximum load (2000 objects and 500 sites), the proposed technique NPAM saved approximately 50 seconds of termination time then the second fastest algorithm (Greedy).

Superiority of execution time comes at the cost of loss in solution quality. However, NPAM showed high solution quality. First, we observe the effects of system capacity increase. 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

Table 6.1: Running time in sec.

Problem Size	Greedy	GRA	Aε-Star	NPAM
M= 500, N= 1350	**81.69**	117.60	110.46	90.09
M= 500, N= 1400	98.28	127.89	127.89	**95.34**
M= 500, N= 1450	122.43	139.02	139.02	**98.91**
M= 500, N= 1500	134.61	148.47	155.40	**104.37**
M= 500, N= 1550	146.58	168.84	169.47	**105.63**
M= 500, N= 2000	152.25	177.66	189.21	**108.57**

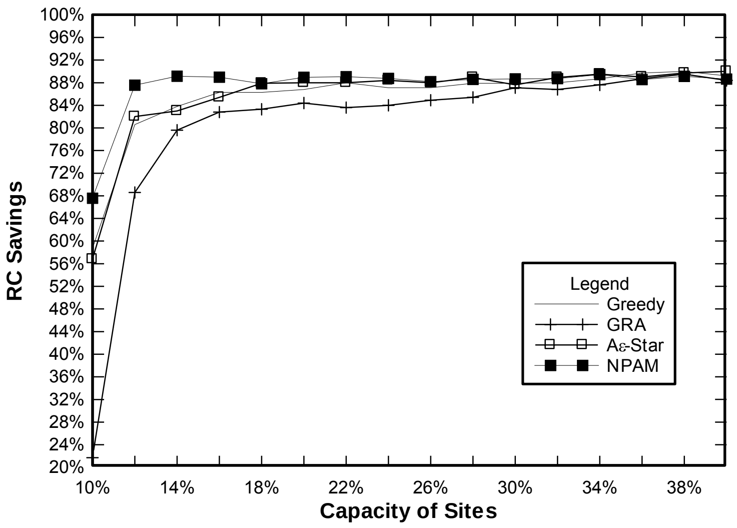

Figure 6.2: RC savings vs. System Capacity (*N* = 2000, *M* = 500, *U* = 5%).

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. This is observable in Figure 6.2, which shows the performance of the algorithms. Greedy and NPAM 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 although performed the worst, but observably gained the most RC savings (35%) followed by Greedy with 29%. Further experiments with

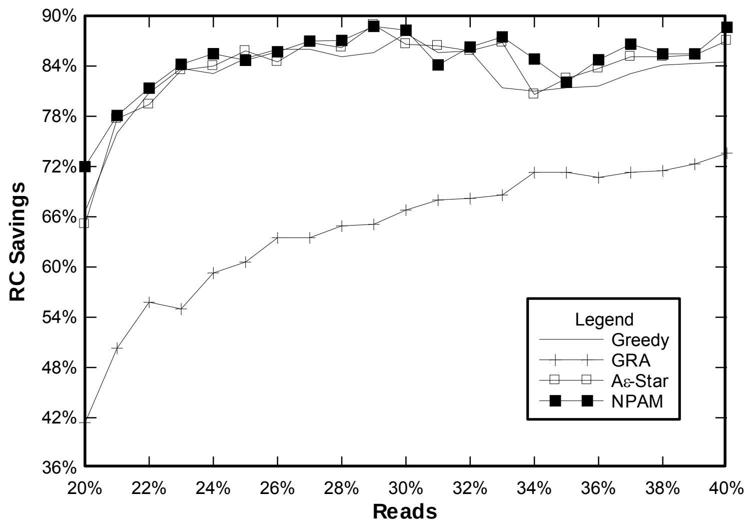

Figure 6.3: RC savings vs. Reads (*N* = 2000, *M* = 500, C = 45%).

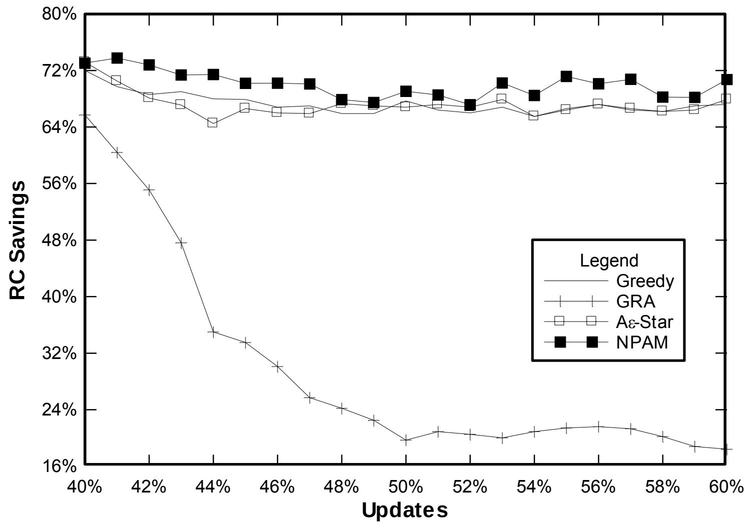

Figure 6.4: RC savings vs. Updates (*N* = 2000, *M* = 500, *C* = 60%).

various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in capacity from 10% to 17%, resulted in 4 times (on average) more replicas for all the

Table 6.2: Average savings in percentage.

Problem Size	Greedy	GRA	Aε-Star	NPAM
N=150, *M*=20 [*C*=20%,*U*=25%]	70.46	69.74	74.62	**75.70**
N=200, *M*=50 [*C*=20%,*U*=20%]	73.94	70.18	77.42	**78.43**
N=300, *M*=50 [*C*=25%,*U*=5%]	70.01	64.29	70.33	**82.25**
N=300, *M*=60 [*C*=35%,*U*=5%]	71.66	65.94	72.01	**74.43**
N=400, *M*=100 [*C*=25%,*U*=25%]	67.40	62.07	71.26	**73.89**
N=500, *M*=100 [*C*=30%,*U*=35%]	66.15	61.62	71.50	**75.45**
N=800, *M*=200 [*C*=25%,*U*=15%]	67.46	65.91	70.15	**73.68**
N=1000, *M*=300 [*C*=25%,*U*=35%]	69.10	64.08	70.01	**72.45**
N=1500, *M*=400 [*C*=35%,*U*=50%]	70.59	63.49	70.51	**74.01**
N=2000, *M*=500 [*C*=10%,*U*=60%]	67.03	63.37	72.16	**73.15**

algorithms.

Next, we observe the effects of increase in the read and update (write) frequencies. Since these two parameters are complementary to each other, we 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 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. The plots in Figures 6.3 and 6.4 show the results of read and update frequencies, respectively. A clear classification can be made between the algorithms. Aε-Star, Greedy and NPAM incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 89%. GRA gained the least of the RC savings of up to 67%. To understand why there is such a gap in the performance between the algorithms, we recall from [78] that GRA specifically depends on the initial population of the candidate solution. Moreover, GRA maintains a localized network perception. 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, Aε-Star, Greedy and NPAM never tend to deviate from their global view of the problem domain.

In summary, Table 6.2 shows the quality of the solution in terms of RC percentage for 10 problem instances (randomly chosen), each being a combination of various numbers of sites and objects, with varying storage capacity and update ratio. For each row, the best result is indicated in bold. The proposed NPAM steals the show in the context of solution quality, but Aε-Star and Greedy do indeed give a good competition, with savings within a range of 7%-10% of NPAM.

6.4 Concluding Remarks

Manual mirroring of data objects is a tedious and time consuming operation. This study proposed a game theoretical N+1st price auction mechanism (NPAM) for fine-grained data replication in large-scale distributed computing systems such as the Internet. NPAM is a protocol for automatic replication and migration of objects in response to demand changes. NPAM aims to place objects in the proximity of a majority of requests while ensuring that no hosts become overloaded.

NPAM allows agents to compete for the scarce memory space at sites so that they can acquire the rights to place replicas. To cater for the possibility of cartel type behavior of the agents, NPAM uses N+1st price protocol. This leaves the agents with no option, then to report truthful valuations of the objects that they represent.

NPAM was compared against some well-known techniques, such as: greedy, branch and bound and genetic algorithms. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The experimental setup was designed to mimic a large-scale distributed computing system (the Internet), by using several Internet topology generators and World Cup Soccer 1998 web server access logs. The experimental results revealed that NPAM outperformed the three widely cited and powerful techniques in both the execution time and solution quality. In summary, NPAM exhibited 7%-10% better solution quality and 10%-30% savings in the algorithm termination timings.

CHAPTER 7

A UTILITY MAXIMIZING GAME THEORETICAL REPLICA PLACEMENT TECHNIQUE

A utility maximizing game theoretical replica placement technique termed as *NCOR* (**n**on-**co**operative game theoretical **r**eplica allocation technique) to reduce user perceived Web access delays is proposed in the subsequent text. *NCOR* uses distributed agents that because of their local knowledge act in a self-interested manner in order to enhance the performance of the servers that they represent. This can lead to some performance gains for some servers but has the potential to negatively impact the overall system's performance. *NCOR* uses an effective cost model to guarantee the overall system performance gain despite the self-interested actions of these agents. With spontaneous and non-deterministic strategies, the system can exhibit Nash equilibrium. However, that may or may not guaranteed system-wide performance at a given time. Furthermore, their can be multiple Nash equilibria, making it difficult to decide which one is the best. Instead, we use the notion of pure Nash equilibrium, which if achieved is guaranteed to ensure stable optimal performance. Pure Nash equilibrium can be only achieved by deterministic strategies. In general, the existence of a pure Nash equilibrium is remarkably hard to achieve; however, we prove the existence of such equilibrium in *NCOR*.

7.1 Introduction

A number of techniques for object-based Web content replication have been proposed with the underlying assumption that servers cooperate with one another in order to layout a replica schema that optimizes the overall system performance. For instance, almost all content distribution networks (CDNs) related replica allocation methods ([41], [46]) rely on a centralized decision making body which optimizes a given objective (such as to reduce the communication cost) regardless of the costs incurred by each server [54]. In reality, servers aim at maximizing their own benefits, possibly at the expense of the global optimal [22].

To study this self-interested behavior, we make use of game theoretical techniques and abstract the Web (or large scale distributed computing system) as an agent based model. Each server in the system is represented by an agent which is a computational entity that is capable of autonomous behavior in the sense of being aware of the options available to it when faced with a decision making task related to its domain of interest [58]. These agents are motivated by their individual interests and compete in a **n**on-**co**operative **r**eplica allocation game (*NCOR*). In *NCOR* each agent has two possible actions for each object. If an access is made to an object that is located at a nearby server, then the agent is better off redirecting the request to that server. On the other hand if the object is located at a far off server, then the agent is better off replicating that object.

The goal of this study is to see whether these self-interested agents in *NCOR*, can layout replica schemas that converge to global optimum solution(s) targeted towards reducing the communication cost induced by accessing the objects. Using game theory, we show that in the worst-case scenario, the system as a whole resides in a social optimum domain, *i.e.*, the solution quality can never be worse than a pareto-optimal solution. This social optimum domain is used as the basis to prove that *NCOR* indeed converges to global optimum solution(s) that conform to pure Nash equilibrium(s) when the self-interested agents play deterministic strategies according to *NCOR*'s cost model. Pure Nash equilibrium is different from the classical Nash equilibrium in the sense that the former results when the strategies played are deterministic, while the later results when the strategies played are non-deterministic. Also, a system will achieve a global optimum solution throughout the lifespan of the system once such a pure Nash equilibrium is achieved. This is certainly not the case when a system exhibits a classical Nash equilibrium, for the simple reason that there could be multiple Nash equilibria, making it difficult to decide which one is the best. An elaborate discussion on these two types of Nash equilibria, their properties and differences can be found in [32] and [112].

7.2 Non-cooperative Replica Allocation Game

Before we discuss the exact game structure of *NCOR* (the non-cooperative replica allocation game), it is essential to lay down the basis of *NCOR*. We start by defining:

7.2.1 Preliminaries for the NCOR

Definition 7.1 (Feasible Strategies): *An agent i's strategy is termed feasible, φ_i, when the two constraints of the data replication problem (storage and no de-allocation of the primary copy) are met before a decision to replicate an object O_k can be undertaken.*

Of all these possibly infinity many feasible strategies, let $\varsigma_i \in \varphi_i$ be a strategy chosen by an agent *i*, where ς_i=1 means object is replicated and ς_i=0 means it is not. (Note that ς_i only focus on a specific object O_k. Therefore it is not necessary to write ς_i as $\varsigma_{i,k}$ or any other notation that would differentiate any two objects.) Since each agent chooses $\varsigma_i \in \varphi_i$ independently (keeping both the constraints at par), we can look at the replication of each object O_k as a separate game, and combine the pure Nash equilibrium of these games to obtain a pure Nash equilibrium of the multi-object game, *NCOR*. (This argument would become clearer when Definition 7.3 and Lemma 7.1 are reviewed.)

Definition 7.2 (Strategy Profile) [96]**:** *A strategy profile $\varsigma=(\varsigma_1,\ldots,\varsigma_M)$ is a set of strategies for each agent which fully specifies all of its actions. A strategy profile must include one and only one strategy for every agent.*

For convenience we can also write ς as $(\varsigma_i,\varsigma_{-i})$, where ς_i is the strategy of agent *i*, and ς_{-i} is the set of strategies of all other agents in *NCOR* excluding agent *i*. Given ς one can easily find out which agents have opted to replicate O_k.

Definition 7.3 (Pure Nash Equilibrium) [112]: *A situation in a non-cooperative game in which agents play using a set of deterministic strategies whereby no agent can improve its benefit by changing its strategy unilaterally.*

Lemma 7.1 (Combining Pure Nash Equilibriums) [32]: *If two games are known to have pure Nash equilibriums, then the union of the games is also guaranteed to have a pure Nash equilibrium.* ■

Thus, if we are able to prove that a given ς conforms to a pure Nash equilibrium, then $\cup\varsigma_i$ also conforms to a pure Nash equilibrium. Conversely, if $\chi(\varsigma)$ is the cost function associated with ς, then $\Sigma\chi(\varsigma_i)$ over all *N* objects is the cost associated with $\cup\varsigma_i$. Based on this, we can give a formal mathematical definition of pure Nash equilibrium:

Definition 7.4 (Pure Nash Equilibrium (Mathematically)): *Let (ς,χ) be a game, where ς is the set of strategy profiles and χ is the cost function. When each agent i plays ς_i then agent i incurs a cost $\chi_i(\varsigma)=\chi_i(\varsigma_{1,\ldots,}\varsigma_M)$. ς^* is pure Nash equilibrium if for any deviation ς_i by an agent i is not beneficial, that is $\chi_i(\varsigma_i,\varsigma_{-i}^*)\leq\chi_i(\varsigma_i^*,\varsigma_{-i}^*)$.*

Definition 7.5 (Stability of a Pure Nash Equilibrium) [112]: *Equilibrium is* *stable if an infinitesimal small change in the strategy of one agent leads to a situation where the following hold:*

(a): The agent who did not change has no better strategy in the new circumstance.

(b): The agent who did change is now playing with a strictly worse strategy.

It is also important that we accentuate on the difference between ς and R_k (replica schema of O_k). If we are given R_k, we only know which servers hold a copy of O_k, but if ς is given, we can also find out which agents have not opted to replicate O_k along with their corresponding cost functions.

Definition 7.6 (Replica Schema): *A replica schema, R_k, for object O_k is the set of servers that replicates O_k.*

7.2.2 NCOR Structure and Mechanism

We now proceed with describing the game structure of *NCOR*.

7.2.2.1 The Setup

The Web (or large scale distributed computing system) is considered, where each server is represented by an agent, *i.e.*, *NCOR* contains *M* agents. Although *NCOR* is non-cooperative in nature, yet there is no information hiding. That is, the network topology, the size of the object and location of replicas are all public knowledge. The only information that is private to each agent is the frequency of reads and writes for each object from its server.

7.2.2.2 Cost Model

We first concentrate on deriving the cost model for a single object. This will be expanded later on to fully encapsulate the multi-object replica placement problem.

Let φ_i be the set of feasible strategies for an agent i. For O_k, the agent chooses a strategy $\varsigma_i \in \varphi_i$ that describes its desire to replicate or otherwise. Thus, given a strategy profile ς, we say that an agent i incurs a cost $\chi_i(\varsigma)$ if it considers replicating object O_k. This cost is given as:

$$\chi_i(\varsigma)=\sum_{k\in\varsigma_i}\left[w_i^k o_k\left(\sum_{\forall j\in R_k, i\neq j} c\left(P_k,j\right)\right)\right]+\sum_{k\notin\varsigma_i}\left[r_i^k o_k c\left(i,NN_i^k\right)+w_i^k o_k c\left(i,P_k\right)\right], \tag{7.1}$$

which implies that if an agent replicates O_k, then the cost incurred due to reads is $0 = r_i^k o_k \mathrm{c}(i,NN_i^k)$ since $NN_i^k = i$. The cost incurred due to local writes (or updates) is equal to zero since the copy resides locally, but whenever O_k is updated anywhere in the network, agent i has to continuously update O_k's contents locally as well. Therefore, the aggregate cost of writes is equivalent to $w_i^k o_k \Sigma_{\forall(j\in Rk),\, i\neq j}\, c(P_k,j)$. On the other hand if an agent does not replicate O_k, then the cost incurred due to reads is equal to $r_i^k o_k \mathrm{c}(i,NN_i^k)$, and the cost incurred due to writes is equal to $w_i^k o_k \mathrm{c}(i,P_k)$ since it only has to send the update to the primary server which then broadcasts the update based on R_k to the agents who have replicated the object.

Equation 7.1 captures the dilemma faced by an agent i when considering replicating O_k. If agent i replicates O_k then it brings down the read cost to zero, but now it has to keep the contents of O_k up to date. If agent i does not replicate O_k, then it reduces the overhead of keeping the contents up to date, but now it has to redirect the read requests to the nearest neighborhood server which holds a copy of O_k. Keeping these cost considerations in mind, for an object O_k each agent i has two strategies: (0) not to replicate or (1) to replicate; allowing us to rewrite Equation 7.1 in a visually appealing form:

$$\chi_i(\varsigma)=\varsigma_i\left[w_i^k o_k\left(\sum_{\forall j\in R_k, i\neq j} c\left(P_k,j\right)\right)\right]+\left(1-\varsigma_i\right)\left[r_i^k o_k c\left(i,NN_i^k\right)+w_i^k o_k c\left(i,P_k\right)\right]. \qquad (7.2)$$

7.2.2.3 Discussion on Cost Model

Each agent i's cost to replicate an O_k (or otherwise) sturdily relies on the access (both read and write) frequencies, the replica locations, and the size of O_k (o_k). Essentially, *NCOR* starts with a given (possibly a random) replica schema, and evolves it into a replica schema that exhibits pure Nash equilibrium as each agent alters its strategy so as to minimize its cost. That is, a pure Nash equilibrium ($\varsigma^*_i,\varsigma^*_{-i}$) for *NCOR* identifies a replica schema R_k such that $\forall i\in M$, $i\in R_k$ if and only if $\varsigma^*_i=1$. Recall that there can be infinitely many feasible strategies, which in turn means that there can be infinitely many replica schemas that are identifiable by a pure Nash equilibrium. Let Є represent the set of all possible pure Nash equilibrium replica schemas and we say:

Definition 7.7 (Pure Nash Equilibrium Replica Schema): *A replica schema* *belongs to the set of pure Nash equilibrium replica schemas $R_k \in Є$ if and only if each agent $i \in M$ chooses a feasible strategy $\varsigma_i \in \varphi_i$ such that when each agent i plays ς_i, it cannot improve its cost by changing its strategy unilaterally, that is $\chi_i(\varsigma_i, \varsigma_{-i}^*) \leq \chi_i(\varsigma_i^*, \varsigma_{-i}^*)$, where ς^* is a pure Nash equilibrium.*

Keeping Definition 7.7 in mind, for *NCOR* we can straightforwardly deduce the following:

$R_k \in Є$ if and only if:

(a) $\forall i \in M$, $\exists j \in R_k$ such that $c(i,j)$ is minimum. (7.3)

(b) $\forall j \in R_k$, $\nexists\, j' \in R_k$ such that $c(j,j') < c(i,j)$. (7.4)

We observe that for an object O_k's replica schema to be in a state of pure Nash equilibrium, each agent i has placed O_k's replica at a server that incurs minimum possible communication cost from S^i. (That is, if the replica is not placed at i, then it is replicated at a server j which has the minimum cost of communication from S^i, compared to any other server in the system.) On the other hand if agent i, has already replicated object O_k, then there is no benefit for agent i to drop the replica since the location incurs a minimal communication cost to at least one server (which holds the replica). Note that what we have just discussed above (Equations 7.3 and 7.4) is equivalent to the two conditions ((a) and (b), respectively) of equilibrium stability as stated in Definition 7.5. With this said, we now expand this single object replica allocation cost model to the multi-object data replication problem. First, let us see what is the cost incurred by the society (all M agents) as a whole.

Definition 7.8 (Social Optimum) [96]**:** *The maximum net benefits for everybody in society, regardless of who enjoys the benefits.*

Social optimum is the analogous concept of optimum resource allocation [112]. In this study since the resources are replicas, we can say the social optimum is equivalent to the optimum replica allocation, and we note that:

Definition 7.9 (Pareto Optimum) [96]**:** *A pareto optimum is a situation in which it is not possible to make any one agent better off without making some other agent worse off.*

Lemma 7.2 (A Condition for Pareto Optimum) [96]**:** *A pareto optimum is not possible unless the net benefits for every agent in society are maximized.* ■

In an ideal price based competitive economy, achieving a social (or pareto) optimum is no big deal [96]. Every agent maximizes its private benefit, but since every agent pays for any benefits it receives, and bears only the corresponding costs, the result of this private benefit maximization is that social net benefits are maximized. However, when pricing is not involved (as is the case in *NCOR*), it is no longer trivial to guarantee social optimum. We write the social cost for *NCOR* as:

$$\chi(\varsigma)=\sum_{i=1}^{M}\chi_i(\varsigma)\,. \tag{7.5}$$

Refining Equation 7.5 using the definition of social optimum we say:

$$\chi(\varsigma_{\min}) = \min_{\varsigma} \chi(\varsigma). \tag{7.6}$$

Equation 7.6 encapsulates the notion of cooperation among all agents to layout a replica schema that incurs minimum communication cost. But the agents are self-interested, hence we use $\chi(\varsigma_{\min})$ as an important measure for the solution quality. What the agents are trying to achieve in conjunction to $\chi(\varsigma_{\min})$ is:

$$\chi(\varsigma) = \text{minimize} \sum_{i=1}^{M} \chi_i(\varsigma). \tag{7.7}$$

Using Lemma 7.1 we say that:

$$\chi(\varsigma) = \text{minimize} \sum_{k=1}^{N} \sum_{i=1}^{M} \chi_i(\varsigma). \tag{7.8}$$

Expanding Equation 7.8 using Equation 7.1 we obtain:

$$\chi(\varsigma) = \text{minimize} \sum_{k=1}^{N} \sum_{i=1}^{M} \left[\left[w_i^k o_k \left(\sum_{\forall j \in R_k, i \neq j} c\left(P_k, j\right) \right) \right] + \left[r_i^k o_k c\left(i, NN_i^k\right) + w_i^k o_k c\left(i, P_k\right) \right] \right]. \tag{7.9}$$

Thus, the pure Nash equilibrium in *NCOR may* exist when over the set of all objects *N*; all *M* agents maximize their benefits (by minimizing the communication costs). A closer look at Equation 7.9 reveals that it is nothing more than the minimization problem described by Equation 3.3. Hence, the following holds:

Theorem 7.1 (Equivalence): *The data replication problem and the non-cooperative replica allocation game are equivalent and have the same objective.* ■

Based on the above discussion we describe the procedure for *NCOR* in Figure 7.1.

7.2.2.4 Description of NCOR Procedure

We maintain a list L^i at each server. This list contains all the objects that can be replicated by agent i onto server S^i. (In other words L^i represents the set of feasible strategies.) We can obtain this list by examining the two constraints of the data replication problem. List L^i would contain all the objects that have their size less then the total available space b^i. Moreover, if server S^i is the primary host of some object O_k, then O_k would not be in L^i. We also maintain a list LS containing all servers that can replicate an object, *i.e.*, $S^i \in LS$ if $L^i \neq$NULL. The algorithm works iteratively. In each step the servers calculate the cost of replicating an object O_k using Equation 7.2 (Line 04). This cost is compared to the current cost incurred by the server. If this new cost is less or equal to the current cost, then the server opts to replicate that object. After a decision of replication is taken, each server updates the server storage capacity and the nearest neighbor list (Lines 8 and 9). Servers also evict the object from the list L^i since a decision on it has already been undertaken (Line 12). This procedure continues till the list L^i becomes empty. When L^i becomes empty, a message is sent to the moderator M (which is a control thread) to evict the server from the game since it is no longer able to undertake any further decisions (Line 14). We would like to clarify that the list L^i is dynamically update in accordance to with the changes of server capacity. For example, if by replicating an object a server exhausts all of its storage capacity, then M dynamically adjusts L^i to empty.

The *NCOR* Procedure

```
Initialize:
$LS, L^i, \chi^*_i(\varsigma)=\infty, M, \varsigma$=NULL
01 WHILE $LS \neq$ NULL DO
02   PARFOR each $S^i \in$LS DO
03         FOR each $O_k \in L^i$ DO
04            Compute $\chi_i(\varsigma)=\min\{\chi_i(\varsigma)|\varsigma_i=1, \chi_i(\varsigma)|\varsigma_i=0\}$;          /* Eq. 7.2 */
05               IF $\chi_i(\varsigma) \leq \chi^*_i(\varsigma)$ THEN
06                  $\chi^*_i(\varsigma)=\chi_i(\varsigma)$;                    /* Update current best cost */
07                  $\varsigma_i=1$;                          /* Replicate object $O_k$ */
08                  $b^i=b^i - o_k$;                     /* Update capacity */
09                  Update $NN_k^i$;                 /* Update the nearest neighbors */
10               ELSE
11                  $\varsigma_i=0$;                          /* Do not replicate object $O_k$ */
12            $L^i = L^i - O_k$;                      /* Update the list*/
13            IF $L^i$ = NULL THEN
14                SEND info to $M$ to update $LS = LS - S^i$;
15         ENDFOR
16   ENDPARFOR                                /*Social cost achieved Equation 7.5 */
17 ENDWHILE                          /* Pure Nash equilibrium achieved Th. 7.2 and 7.3 */
```

Figure 7.1: The Pseudo-code for *NCOR* Procedure.

Theorem 7.2 (Existence of Pure Nash Equilibrium): *A pure Nash equilibrium exists for the self-interested agents, if they play according to the cost model of single object NCOR.*

Proof: Let M denoted the set of agents in the system, where each agent represents a server. Let $c(i,NN_i^k)$ represent the cost of assessing object O_k from server S^i to replicated at the nearest server from NN_i^k. Let M' represent the set of servers for which a server S^i incurs the minimum communication cost for all servers $m \in M'$, *i.e.*, $M' = \{m \mid c(i,m) \leq c(i,NN_i^k)\}$. Essentially, *NCOR* chooses a server $m \in M'$ such that $c(i,m) \leq c(i,NN_i^k)\ \forall\ i \in M$ to hold the replica. After allocating a replica at m, it is removed along with all servers $m \in M'$ from M. This is done because no servers (m) has incentive to replicate O_k since it can access m's replica at a lower or equal cost than

NN_i^k's replica. *NCOR* iteratively chooses a server m till $M = \varnothing$. Again, since at each iteration m is the remaining server with minimum $c(i,m)$, no other server can be selected to replicate O_k such that $c(i,NN_i^k) \leq c(i,m)$. Hence, no agent can gain benefit by unilaterally opting to replicate an object without disturbing the equilibrium. ■

Theorem 7.3 (NCOR Pure Nash Equilibrium): *A pure Nash equilibrium exists for the multi-object NCOR.*

Proof: Follows from Lemma 7.1 and Theorem 7.2. ■

7.3 Experimental Comparative Analysis

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory, using the same experimental infrastructure as described in Chapter 3. The experimental evaluations were targeted to benchmark the placement policies. *NCOR* was implemented using Ada and Ada GNAT's distributed systems annex GLADE [98].

The solution quality was measured in terms of network communication cost (OTC percentage) that was saved under the replica scheme found by the replica allocation methods, compared to the initial one, *i.e.*, when only primary copies exists.

7.3.1 Comparative Algorithms

For comparisons, we chose three types of replica allocation methods. To provide a fair comparison, the assumptions and system parameters were kept the same in all the methods. For the data replication problem, the non-game theoretical techniques proposed in [57], [75], [78] and [100] are the only ones that address the problem domain similar to ours. We select from [100] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [75]); thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [57] (the efficient branch and bound based technique Aε-Star) and [78] (the genetic algorithm based method GRA) are also among the chosen techniques for comparisons.

7.3.2 Comparative Analysis

First, we observe the effects of increase in storage 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. This is observable in Figure 7.2, which shows the performance of the algorithms. The performance between all approaches except GRA was within 15% of each other. *NCOR* and Greedy showed an immediate initial increase (the point after which further replicating objects is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA performed the worst, but observably gained the most OTC savings (49%) with various read/write ratios (0.90, 0.80, and 0.70) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in

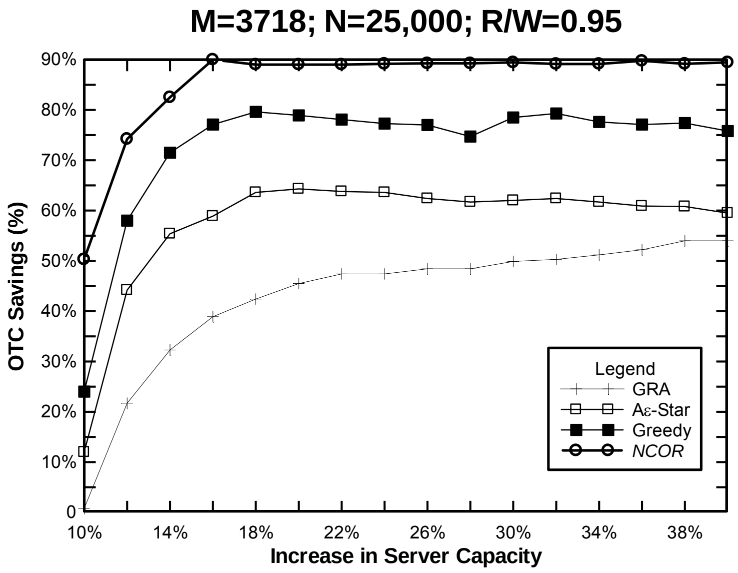

Figure 7.2: OTC savings versus capacity.

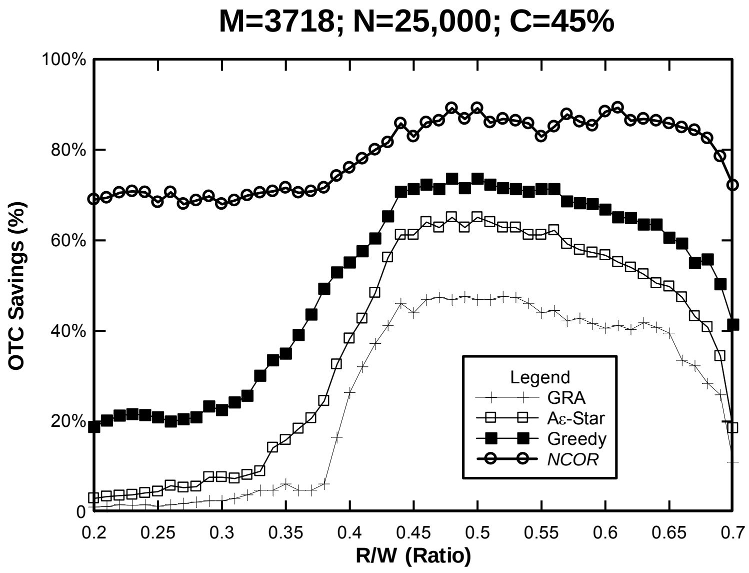

Figure 7.3: OTC savings versus read/write ratio.

capacity from 10% to 18%, resulted in 4 times more replicas for all the algorithms.

Next, we observe the effects of increase in the read and write frequencies. Since these two parameters are complementary to each other, we describe them together. To observe the system utilization with varying read/write frequencies, we kept the number

Table 7.1: Running time of the replica placement methods in seconds for small problem instances [*C*=20%, *R/W*=0.45]

Problem Size	Greedy	GRA	Aε-Star	*NCOR*
M=200, *N*=500	84.13	111.19	116.61	**37.03**
M=200, *N*=1000	91.90	115.68	123.56	**43.34**
M=200, *N*=1500	93.91	121.21	136.62	**51.85**
M=300, *N*=500	114.28	152.30	168.93	**58.81**
M=300, *N*=1000	131.00	150.04	178.59	**65.19**
M=300, *N*=1500	162.25	178.30	215.68	**70.98**
M=400, *N*=500	151.68	184.95	238.52	**76.06**
M=400, *N*=1000	161.58	202.17	284.00	**88.27**
M=400, *N*=1500	169.29	245.31	324.75	**95.55**

Table 7.2: Running time of the replica placement methods in seconds for large problem instances [*C*=45%, *R/W*=0.85]

Problem Size	Greedy	GRA	Aε-Star	*NCOR*
M=2500, *N*=15,000	310.14	491.00	399.63	**188.95**
M=2500, *N*=20,000	330.75	563.25	442.66	**205.45**
M=2500, *N*=25,000	357.74	570.02	465.52	**233.14**
M=3000, *N*=15,000	452.22	671.68	494.60	**286.35**
M=3000, *N*=20,000	467.65	726.75	498.66	**290.31**
M=3000, *N*=25,000	469.86	791.26	537.56	**303.85**
M=3718, *N*=15,000	613.27	883.71	753.87	**372.66**
M=3718, *N*=20,000	630.39	904.20	774.31	**390.38**
M=3718, *N*=25,000	646.98	932.38	882.43	**401.88**

of servers and objects 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 in the system requires the replicas be placed as close as to the primary server 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. The plot in Figure 7.3 shows the results of read/write ratio against the OTC savings. A clear classification can be made between the followed by Greedy with 44%. Further experiments algorithms. *NCOR*, Aε-Star and Greedy incorporate the increase in the number of reads by replicating more objects and thus savings increased up to 88%, while GRA gained the least of the OTC savings of up to 42%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial selection of gene population (for details see [78]). Moreover, GRA maintains a localized network perception. 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, *NCOR*, Aε-Star and Greedy never tend to deviate from their global view of the problem.

Lastly, we compare the termination time of the algorithms. Various problem instances were recorded with *C*=20%, 45% and *R/W*=0.45, 0.85. The entries in Tables 7.1 and 7.2 made bold represent the fastest time recorded over the problem instance. *NCOR* terminated faster than all the other techniques, followed by Greedy, Aε-Star and GRA.

7.4 Concluding Remarks

The replica placement problem recognizes the need of simultaneous allocation of replicas and results in an optimization problem in which the communication cost is reduced subject to the availability of storage and no de-allocation of primary copy. This is particularly useful when data is replicated in a large scale distributed computing system such as the Web, which requires the incorporation of “read from the nearest” and “update through the primary server policies.

We proposed a utility maximizing game theoretical technique, in which the Web was abstracted as an agent based where each agent represented a server. A detailed discussion revealed that in a realistic system, agents have no incentive to cooperate and achieve a social optimum. To this end, we proposed a non-cooperative replica allocation game (*NCOR*), in which agents competed to host the replicas of different objects in a selfish manner, and *NCOR* exhibited a pure Nash equilibrium. Although in game theory literature there are very rare occurrences of pure Nash equilibrium, yet we showed that if agents play using deterministic selfish strategies then *NCOR* conforms to a pure Nash equilibrium.

In *NCOR* each agent had two possible actions for each object. If an access was made to an object that was located at a nearby server, then the agents was better off redirecting the request to that server. On the other hand if the object was located at a far off server, then the agent was better off replicating that object. Essentially for each object the agent made a binary decision: (0) not to replicate or (1) to replicate.

CHAPTER 8

A DISCRIMINATORY GAME THEORETICAL REPLICA PLACEMENT TECHNIQUE

We propose a unique discriminatory replica placement technique using the concepts of a supergame. The supergame allows the agents who represent the data objects to continuously compete for the limited available server memory space, so as to acquire the rights to place data objects at the servers. At any given instance in time, the supergame is represented by a game which is a collection of subgames, played concurrently at each server in the system. We derive a resource allocation mechanism which acts as a platform at the subgame level for the agents to compete. This approach allows us to transparently monitor the actions of the agents, who in a non-cooperative environment strategically place the data objects to reduce the user access time, latency, which in turn adds reliability and fault-tolerance to the system. We show that this mechanism exhibits Nash equilibrium at the subgame level which in turn conforms to games and supergame Nash equilibrium, respectively, guaranteeing the entire system to be in a continuous self-evolving and self-repairing mode.

8.1 Introductory Remarks

Web replication aims to reduce network traffic, server load, and user-perceived delay by replicating popular content on geographically distributed web servers (sites). Specifically, a replica placement algorithm aims to strategically select replicas (or hosting services) among a set of potential sites such that some objective function is optimized under a given traffic pattern.

One might argue that the ever decreasing price of memory renders the optimization or fine tuning of replica placement a “moot point”. Such a conclusion is ill-guided for the following two reasons. First, studies ([4], [13], etc.) have shown that users’ access hit ratio grows in *log*-like fashion as a function of the server memory size. Second, the growth rate of Web content is much higher than the rate with which memory sizes for the servers are likely to grow. The only way to bridge this widening gap is through efficient replica placement and management algorithms.

The Internet can be considered as a large-scale distributed computing system. We abstract this distributed computing system as an agent-based model, where each agent is responsible for (or represents) a data object. Each agent competes in a non-cooperative environment for the limited available storage space at each server so as to acquire the rights to place the data object which they represent. Motivated by their self interests and the fact that the agents do not have a global view of the distributed system, they concentrate on local optimization. In such systems there is no a-priori motivation for cooperation and the agents may manipulate the outcome of the replica placement algorithm (resource allocation mechanism or simply a *mechanism*) in their interests by misreporting critical data such as objects' popularity. To cope with these *selfish* agents, new mechanisms are to be conceived. The goal of a mechanism should be to force the agents not to misreport and always follow the rules.

We use the concepts of game theory to formally specify a mechanism with selfish agents. Game theory assumes that the participating agents have *rational* thoughts that enable them to express their preferences over the set of the possible outcomes of the mechanism. In a mechanism, each agent's benefit or loss is quantified by a function called *valuation*. This function is private information for each agent and is very much possible that if the agents act selfishly, they can misreport their valuations. The mechanism asks the agents to report their valuations, and then it chooses an outcome that maximizes/minimizes a given objective function. Of course the grand problem is to stop the agents from misreporting.

In essence we sculpt the replica placement problem as a *supergame* that is played infinitely during the entire lifespan of the system. In a discrete time instance *t*, the supergame is represented by a *game*, which is the collection of independent *subgames* that are played concurrently at each site of the distributed system. It is in these subgames that the actual mechanism can be seen to operate.

8.2 The Proposed Mechanism

In game theory, usually mechanisms refer to auctions. Mechanisms are used to make allocation and pricing decisions in a competitive environment where all involved parties act strategically in their own best interests. In recent years, many areas of mathematical sciences research started to focus on strategic behavior and, consequently, we are witnessing the use of mechanisms in areas where pure optimization techniques were dominant in the past. For example, in the context of distributed systems, such mechanisms have been applied to the scheduling problems [39], [94], etc.

One has to be careful when incorporating a “one-size-fits-all” mechanism model as a piece of solution to a problem. Most of the mechanisms were developed and analyzed in microeconomic theory abstraction. Thus, assumptions underlying desirable properties of some mechanisms could be oversimplifying or even contradictory to the assumptions underlying a problem that plans to incorporate such mechanisms in its solution.

8.2.1 Discriminatory Mechanism

We limit our analysis to one-shot (single round) mechanisms in which every agent demands a specific entity. Under our replica placement problem formulation we aim to identify a replica schema that effectively minimizes the OTC. We propose a one-shot discriminatory mechanism, where the agents compete for memory space at sites so that they can acquire the rights to place replicas. The mechanism described in this study is called discriminatory because not all winning agents pay the same amount. In essence it works as follows: In a discriminatory mechanism, sealed-bids are sorted from high to low, and rights to the available memory space are awarded at the current highest bid price until the (memory) supply is exhausted. The most important point to remember is that the winning agents can (and usually do) pay different prices.

It is to be noted that in a discriminatory mechanism, an agent always bids below its valuation for the entity [38]. If the agent bids at or above its value, then its payment equals or exceeds its value if it wins, and therefore its expected profit will be zero or negative. Since bids are below the agents' value, the discriminatory mechanism is not a demand reveling mechanism [85].

In a discriminatory mechanism, there is no sequential interaction among agents [85]. Therefore, the mechanism environment is non-cooperative in nature. Agents submit the bids only once. Agents are trading between bidding high and winning for certain and bidding low and benefiting more if the bid wins. In [24] the authors have shown that the discriminatory mechanism is a generalization of the first price sealed-bid auction which is strategically equivalent to the Dutch auction. Unlike in the second price sealed-bid and the English auctions, it is not a dominant strategy for a bidder in the first price sealed-bid auction to bid its valuation for the entity. However, the theoretically optimal bidding strategy in both the first price sealed-bid and the Dutch auctions is the same for any given bidder. Since discriminatory auctions are generalization of the first price sealed-bid auctions, the same argument (about the dominating strategies) holds [40].

8.2.2 Preliminaries

Definition 8.1 (Supergame): *Generally a game in which some simple game is played more than once (often infinitely many times); the simple game is called the* *"stage" game or the "constituent" game – a game repeated infinitely is called a supergame. If Γ represents a game then $\Gamma(\infty)$ represents a supergame.*

Definition 8.2 (Stage game (subgame)): *Frequently it is the case that a game naturally decomposes into smaller games. This is formalized by the notion of stage game (more popularly known as subgames).*

Remarks – We explain this concept using decision trees [85]. Let x be a node which belongs to the set of all the nodes, X, in a tree, K, and let K_x be the subtree of K rising at x. If it is the case that ever information set of Γ either is completely contained in K_x or is disjoint from K_x, then the restriction of Γ to K_x constitutes a game of its own, to be called subgame Γ_x starting at x. This decomposition also affects strategies. Let b represent the strategy set for any player i, then the strategy combination b decomposes into a pair (b_{-x}, b_x) where b_x is a strategy combination in Γ_x and b_{-x} is a strategy combination for the remaining part of the game (the truncated game). If it is known that b_x will be played in Γ_x, then, in order to analyze Γ it suffices to analyze the truncated game $\Gamma_{-x}(b_x)$ which results from Γ.

Interestingly, the concept connecting supergame, games, and subgames is the Nash equilibrium.

Definition 8.3 (Nash equilibrium): *If there is a set of strategies with the property that no player can benefit by changing her strategy while the other players* *keep their strategies unchanged, then that set of strategies and the corresponding payoffs constitute the Nash equilibrium.*

Definition 8.4 (Equilibrium path): *For a given (Nash) equilibrium an information set is on the equilibrium path if it will be reached with positive probability when the game is played according to the equilibrium strategies.*

Lemma 8.1 ([40])**:** *Nash equilibrium only depends upon subgame strategy profiles played along the equilibrium path.* ■

Theorem 8.1 ([38]): *In Nash equilibrium each player's repeated game (supergame) strategy need only be optimal along the equilibrium path.* ■

Remarks — In essence Definitions 8.3 and 8.4 and Lemma 8.1 propose that if a game Γ is in Nash equilibrium, it is only so because all subgames Γ_x are in Nash equilibrium. Extending the same concept, Theorem 8.1 asserts that Nash equilibrium can be reached in a supergame via the equilibrium path followed by games. Recall that a supergame is an infinite play of games. In summary, if all the subgames are in Nash equilibrium, the corresponding game that encapsulates the subgames is also in Nash equilibrium and so is the supergame which is the collection of infinite games.

8.3 Mechanism Applied to the Replica Placement Problem

Form the discussion above, we choose the following line of action.

1. Define the replica placement problem as a supergame.
2. Define an instance of the supergame as a game.
3. Split the game into concurrently played subgames. Each identical to each other in terms of:
    a. *Form:* A discriminatory mechanism.
    b. *Valuation:* Obtainable via the system parameters.
    c. *Information:* Independent of any other subgame.
4. Establish the fact that subgames conform to Nash equilibrium provided agents play optimally.
5. Use Lemma 8.1 to establish that the entire game at instance *t* is in Nash equilibrium.
6. Use Theorem 8.1 to establish that the entire supergame is in Nash equilibrium.

8.3.1 Supergame

A supergame $\Gamma(\infty)$ is defined as a mechanism that is played infinitely during the lifespan of the distributed system under consideration. The supergame allows the agents to compete for memory spaces of the sites. The purpose of a supergame is to keep the system in a self evolving and self repairing mode.

8.3.2 Game

At any given instance t, a game Γ is played. It is to be noted that the sole purpose of defining a game is to observe the solution quality of the replica placements at a given instance t [78].

8.3.3 Subgame

A game is split into M concurrently played subgames. Each of these subgames take place at a particular site i. Each agent k competes through bidding for memory at a site i.

8.3.3.1 Form

Each site i has a finite amount of space s^i, and available space b^i. It is for this available space b^i that the agents compete. In one-shot all the participating agents submit their bids for the available space. All the bids are sorted in descending order and the first n agents are awarded the rights to place their objects onto site i. Recall that each agent represents an object of size o_k. Therefore, the decision of the first n agents solely depends upon $\sum_{k=1}^{n} o_k \le b^i, n \le N$. After the decision is made, the first n agents pay their respective bids. This is discriminatory for the following two reasons. First, all the successful agents pay a different amount for their rights to place an object. Second, the payment is in no relation to the size of the object or the available space at site i. The only connection that the payments have is the benefit that the object brings if replicated to that site. This benefit is the valuation of an agent for its object k if replicated at site i.

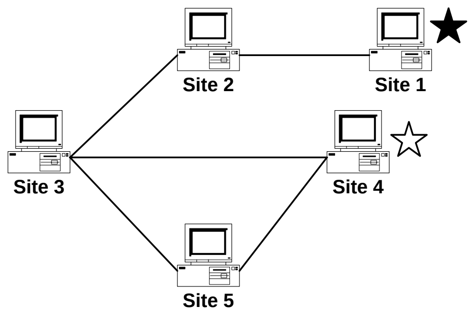

Figure 8.1: The network architecture.

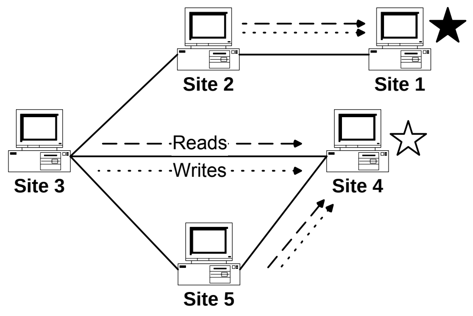

Figure 8.2: Read and write patterns.

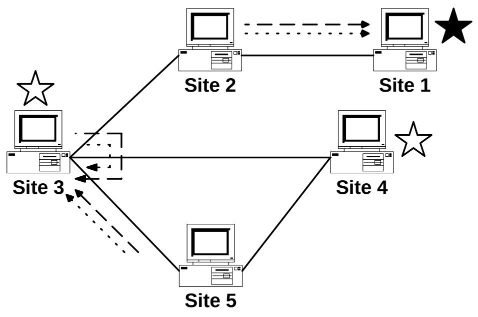

Figure 8.3: Benefits of replication (reads).

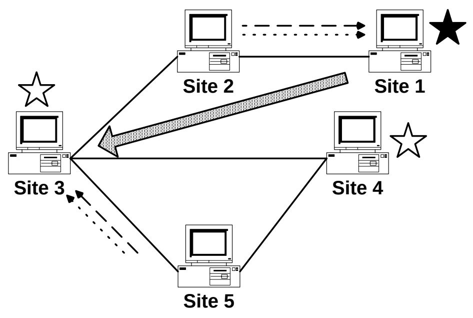

Figure 8.4: Benefits of replication (writes).

We describe this valuation below.

8.3.3.2 Valuation

Each agent *k*'s policy is to place a replica at a site *i*, so that it maximizes its (object's) benefit function. This benefit is equivalent to the savings that the object *k* brings in the total OTC if the object *k* is replicated at site *i*. This benefit is given as:

$$B_k^i = R_k^i - \left(\sum_{x=1}^{M} w_k^x o_k c\left(i, P_k\right) - W_k^i \right). \tag{8.1}$$

We illustrate the notion of benefit associated with an object *k* if it is replicated at site *i*. Figure 8.1 depicts the network with four sites. Site 1 has the primary object represented by ★, while Site 4 has the replica of the same object represented by ☆. If these are the only copies of object *k* available in the network, then the read and write requests are always sent to the nearest neighbors, where Site 4 is the nearest neighbor of itself (Figure 8.2). Now what would be the benefit of replicating object *k* at Site 3? In Figure 8.3, we see that the reads and writes of Site 3 are entertained locally. Moreover, Site 5 can now redirect its request to its newest nearest neighbor, *i.e.*, Site 3. Therefore, the replication of object k at Site 3 clearly reduces the OTC by $RC_k^i = R_k^i + W_k^i$. However (Figure 8.4), this will cause the Site 1 (location of primary object) to repeatedly send updates of object *k* to Site 3. Since the local update is already captured by RC_k^i, the increased aggregate updates are given by:

$$\sum_{x=1}^{M} w_k^x o_k c(i, P_k).$$

From here onwards, for simplicity, we will denote the benefit B_k^i as *v* (valuation). It is to be understood that to differentiate the valuations between agents *k* and j we may denote the valuations as v_k and v_j, respectively.

8.3.3.3 Information

It is clear that the subgames can operate independently of each other. There is no critical information that is required and is withheld from a subgame. For instance, 1) the frequency of reads and writes are obtained locally through the site which hosts the subgame, 2) the information about network architecture is globally available since domains can easily pull such information from the routers using the border gate protocol (BGP) [103], and 3) the locations of the primary sites are also available locally since the agents represent the objects, (*i.e.*, they have to know where they originated from,) etc.

8.3.4 Subgame Nash Equilibrium

To understand the bidding behavior in a discriminatory mechanism, we shall, for simplicity, assume that the agents are ex-ante symmetric. That is, we shall suppose that for all bidders $k = 1,\ldots, N$, $f_k(v) = f(v)$ for all $v \in [0,1]$, where v is the valuation of an agent k for an object, whereas f translates this valuation into something useful, for instance, when bids are required for an object, f can take the form of a bidding function for a valuation v. Note that we only assume that $v \in [0,1]$ for underlying the groundwork for the probabilistic analysis. In reality the valuations are of the form of $v \geq 0$. Clearly, the main difficulty is in determining how the agents, will bid. But note that a rational agent k would prefer to win the right to replicate at a lower price rather than a higher one, agent k would bid low when the others are bidding low and would want to bid higher when the others bid higher. Of course, agent k does not know the bids that the others submit because of the sealed-bid rule. Yet, agent k's optimal bid will depend on how the others bid. Thus, the agents are in a strategic setting in which the optimal action (bid) of each agent depends on the actions of others.

Let us consider the problem of how to bid from the point of view of agent k. Suppose that agent k's value is v_k. Given this value; agent k must submit a sealed-bid, b_k. Because b_k will in general depend on k's value, let's write $b_k(v_k)$ to denote bidder k's bid when his value is v_k. Now, because agent k must be prepared to submit a bid $b_k(v_k)$ for each of his potential values $v \in [0,1]$, we may view agent k's strategy as a bidding function b_k:$[0,1]\rightarrow\Re_+$, mapping each of his values into a (possibly different nonnegative) bid.

Before we discuss payoffs, it will be helpful to focus our attention on a natural class of bidding strategies. It seems very natural to expect that agents with higher values will place higher bids. So, let's restrict attention to strictly increasing bidding functions. Next, because the agents are ex-ante symmetric, it is also natural to suppose that agents with the same value will submit the same bid. With this in mind, we shall focus on finding a strictly increasing β function, $\hat{b}_k$:$[0,1]\rightarrow\Re_+$, that is optimal for each agent to employ, given that all other agents employ his bidding function as well. That is, we wish to find Nash equilibrium in strictly increasing bidding functions.

Now, let us suppose that we find Nash equilibrium given by the strictly increasing bidding function $\hat{b}(\cdot)$. By definition it must be payoff-maximizing for an agent, say k, with value v to bid $\hat{b}(v)$ given that the other agents employ the same bidding function $\hat{b}(\cdot)$.

Remarks — We explain why we assume that all other agents employ the same bidding function $\hat{b}(\cdot)$. Imagine that agent k cannot attend the auction and that he sends a friend to bid for him. The friend knows the equilibrium bidding function $\hat{b}(\cdot)$ (since it is a public knowledge), but does not know agent k's value. Now, if agent k's value is v, agent k would like his friend to submit the bid $\hat{b}$ (v) on his behalf. His friend can do this for him once agent k calls him and tells his value. Clearly, agent k has no incentive to lie to his friend about his value. That is, among all the values $r \in [0,1]$ that agent k with value v can report to his friend, his payoff is maximized by reporting his true value, v, to his friend. This is because reporting the value r results in his friend submitting the bid $\hat{b}$ (r) on his behalf. But if agent k were there himself he would submit the bid $\hat{b}(v)$.

Let us calculate agent k's expected payoff from reporting an arbitrary value, r, to his friend when his value is v, given that all other agents employ the bidding function $\hat{b}(\cdot)$. To calculate this expected payoff, it is necessary to notice just two things. First, agent k will win only when the bid submitted for him is highest. That is, when $\hat{b}(r) > \hat{b}$ (v_j) for all agents $j \neq k$. Because $\hat{b}$ $(\cdot)$ is strictly increasing this occurs precisely when r exceeds the values of all N-1 other agents. Let F denote the distribution function associated with f, the probability that this occurs is $(F(r))^{N\text{-}1}$ which we will denote $F^{N\text{-}}$

$^{1}(r)$. Second, agent *k* pays only when it wins the right to replicate, and pays its bid, $\hat{b}(r)$. Consequently, agent *k*'s expected payoff from reporting the value *r* to his friend when his value is *v*, given that all other bidders employ the bidding function $\hat{b}(\cdot)$, can be written as:

$$u(r,v)=F^{N-1}(r)\left(v-\hat{b}(r)\right). \tag{8.2}$$

Now, as we have already remarked, because $\hat{b}(\cdot)$ is an equilibrium, agent *k*'s expected payoff-maximizing bid when his value is *v* must be $\hat{b}(v)$. Consequently, Equation 8.2 must be maximized when $r = v$, *i.e.*, when agent *k* reports his true value, *v*, to his friend. So, we may differentiate the right-hand side with respect to *r* and set the derivative equal to zero when $r = v$. Differentiating yields:

$$d/dr\left(F^{N-1}(r)\left(v-\hat{b}(r)\right)\right)=\left(N-1\right)F^{N-2}(r)f(r)\left(v-\hat{b}(r)\right)-F^{N-1}(r)\hat{b}'(r)\cdot \tag{8.3}$$

Setting this equal to zero when $r = v$ and rearranging yields:

$$\left(N-1\right)F^{N-2}(v)f(v)\hat{b}(v)+F^{N-1}(v)\hat{b}'(v)=\left(N-1\right)vf(v)F^{N-2}(v). \tag{8.4}$$

Looking closely at the left-hand side of Equation 8.4, we see that is just the derivative of the product $F^{N-1}(v)$ times $\hat{b}(v)$ with respect to *v*. With this observation, we can rewrite Equation 8.4 as:

$$d/dv\left(F^{N-1}(v)\hat{b}(v)\right)=\left(N-1\right)vf(v)F^{N-2}(v). \tag{8.5}$$

Now, because Equation 8.5 must hold for every v, it must be the case that:

$$F^{N-1}(v)b(v) = \left(N-1\right)\int_0^v xf(x)F^{N-2}(x)dx + constant\,. \tag{8.6}$$

Noting that an agent with value zero must bid zero, we conclude that the constant above must be zero.

Hence, it must be the case that:

$$\hat{b}(v) = \frac{N-1}{F^{N-1}(v)}\int_0^v xf(x)F^{N-2}(x)dx\,, \tag{8.7}$$

which can be written as:

$$\hat{b}(v) = \frac{1}{F^{N-1}(v)}\int_0^v xf(x)F^{N-2}(x)dx\,. \tag{8.8}$$

There are two things to notice about the bidding function in Equation 8.8. First, as we has assumed, it is strictly increasing in *v*. Second, it has been uniquely determined. Now since we assumed that each agent is ex-ante in nature, then *F*(*v*) = *v* and *f*(*v*) = 1. Consequently, if there are *N* bidders then each employs the bidding function:

$$\hat{b}(v) \;=\; \frac{1}{v^{N-1}}\int_0^v x dx^{N-1} \tag{8.9}$$

$$= \frac{1}{v^{N-1}}\int_0^v x\left(N-1\right)x^{N-2}dx$$

$$= \left(\frac{N-1}{v^{N-1}}\right)\left(\frac{1}{N}\right)v^N$$

$$= \left(\frac{N-1}{N}\right)v \tag{8.10}$$

Hence, in conclusion, we have proven the following:

Theorem 8.2: *If N agents have independent private values drawn from the*

```
Discriminatory Mechanism
Initialize:
01 $LS$, $L^i$.
02 WHILE $LS \neq$ NULL DO
03   PARFOR each $S^i \in$ LS DO                    /* $M$ subgames */
04       FOR each $k \in O$ DO
05           $B_k$ = compute $(B_k^i \times (N-1)/N)$;       /* Compute benefit */
06           Report $B_k$ to $S_i$ which is stored in array $B$;
07       END FOR
08       Sort array $B$ in descending order.
09     WHILE $b^i \geq 0$
10     $B_k = \text{argmax}_k(B)$;                  /* Choose the best offer */
11     Extract the info from $B_k$ such as $O_k$ and $o_k$;
12     $b^i = b^i - o_k$;                   /* Calculate space and termination condition */
13     Replicate $O_k$;
14     Payment = $B_k$;                     /* Calculate payment */
15     Delete $B_k$ from $B$;                /* Update the list for highest bid */
16     SEND $P^i$ to $S^i$; RECEIVE at $S^i$        /* Agent pays the bid */
17     $L^i = L^i - O_k$;                    /* Update the list */
18     Update $NN^i_{OMAX}$                  /* Update the nearest neighbor list */
19     IF $L^i$ = NULL THEN SEND info to $M$ to update $LS = LS - S^i$;   /* Update the player list */
20     END WHILE
21   ENDPARFOR
22 END WHILE
```

Figure 8.5: Mechanism game at instance *t*.

common distribution, F, then bidding $\hat{b}(v) = (N\text{-}1/N)v$ *whenever one's value is v constitutes Nash equilibrium of the discriminatory mechanism, where the nature of the bids are sealed-bids.* ■

So, each agent *shades* its bid, by bidding less than its valuation. Note that as the number of agents increases, the agents bid more aggressively. Because $F^{N-1}(\cdot)$ is the distribution function of the highest value among an agent's N-1 competitors, the bidding strategy displayed in Theorem 8.2 says that each agent bids the expectation of the second highest agent's value conditional on his value being highest. But, because the agents use the same strictly increasing bidding function, having the highest value is equivalent to having the highest bid and so equivalent to winning the right to replicate.

Theorem 8.3: *If N agents play their bids according to the bidding strategy as: $\hat{b}(v) = (N\text{-}1/N)v$, the corresponding game at instance t and eventually the supergame are in Nash equilibrium.*

Proof: It follows from Lemma 8.1 and Theorem 8.1. ■

We are now ready to present the pseudo-code (Figure 8.5) for a game at instance *t*.

Briefly, we maintain a list L^i at each server. The list contains all the objects that can be replicated at S^i (i.e., the remaining storage capacity b^i is sufficient and the benefit value is positive). We also maintain a list *LS* containing all servers that can replicate an object. In other words, $S^i \in LS$ if and only if $L^i \neq$ NULL. Each player $k \in O$ calculates the benefit function of object (Line 05). The set *O* represents the collection of players that are legible for participation. A player *k* is legible if and only if the benefit function value obtained for site S^i is positive. This is done in order to suppress mediocre bids, which, in turn improves computational complexity. After receiving (Line 06) all the bids, the bid vector is sorted in descending order (Line 08). Now, recursively the rights are assigned to the current highest agent (Line 10) as long as there is available memory (Line 09 and 12). It is to be noted that in each step L^i together with the corresponding nearest server value NN_k^i, are updated accordingly.

The above discussion allows us to deduce the following result about the mechanism.

Theorem 8.4: *In the worst case the mechanism takes $O(N^2 \log N)$ time.*

Proof: The worst case scenario is when each site has sufficient capacity to store all objects. In that case, the PARFOR loop (Line 03) performs N iterations. The most consuming time is to sort the bids in descending order (Line 10). This will take at least of the order of $O(N\log N)$. Hence, we conclude that the worst case running time of the mechanism is $O(N^2\log N)$. ■

8.4 Experimental Comparative Analysis

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory using the same experimental infrastructure as described in Chapter 3. The experimental evaluations were targeted to benchmark the placement policies. The mechanism was implemented using IBM Pthreads.

The solution quality is measured in terms of network communication cost (OTC percentage) that is saved under the replication scheme found by the algorithms, compared to the initial one, i.e., when only primary copies exists.

8.4.1 Comparative Algorithms

For comparisons, we selected five various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The techniques studied include efficient branch-

and-bound based technique (Aε-Star [57]). For fine-grained replication, the algorithms proposed in [75], [78], and [100] are the only ones that address the problem domain similar to ours. We select from [100] the greedy approach (Greedy) for comparison because it is shown to be the best compared with four other approaches (including the proposed technique in [75]); thus, we indirectly compare with four additional approaches as well. Algorithms reported in [58] (Dutch (DA) and English auctions (EA)) and [78] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons.

8.4.2 Comparative Game Analysis

First, we concentrate on observing the improvement brought by the discriminatory mechanism (for short we will refer to it as MECH). To this end we observe the solution quality at the game level. In the post-ceding text we shall discuss the results obtained in the supergame setup.

We study the behavior of the placement techniques when the number of sites increases (Figure 8.6), by setting the number of objects to 2000, while in Figure 8.7, we study the behavior when the number of objects increase, by setting the number of sites to 500. We should note here that the space limitations restricted us to include various other scenarios with varying capacity and update ratio. The plot trends were similar to the ones reported in this article. For the first experiment we fixed C = 30% and U = 65%. We intentionally chose a high workload so as to see if the techniques studied successfully handled the extreme cases. The first observation is that MECH and EA

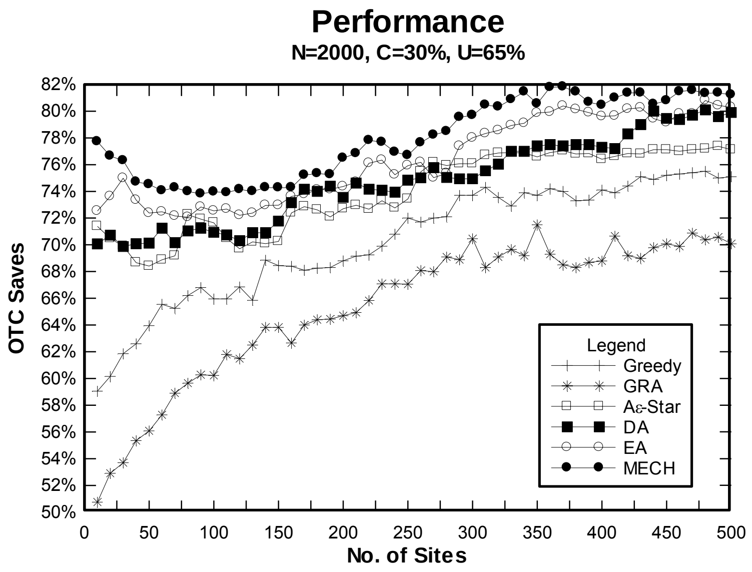

Figure 8.6: OTC savings versus number of sites.

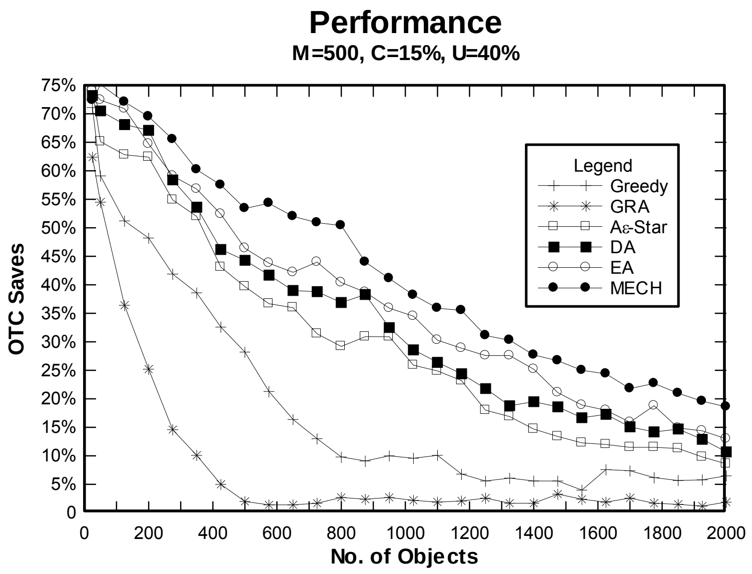

Figure 8.7: OTC savings versus number of objects.

outperformed other techniques by considerable amounts. Second, DA converged to a better solution quality under certain problem instances than EA. This is inline with the general trends of DA. It outperforms EA when the agents are bidding aggressively. Some interesting observations were also recorded, such as, all but GRA and Greedy

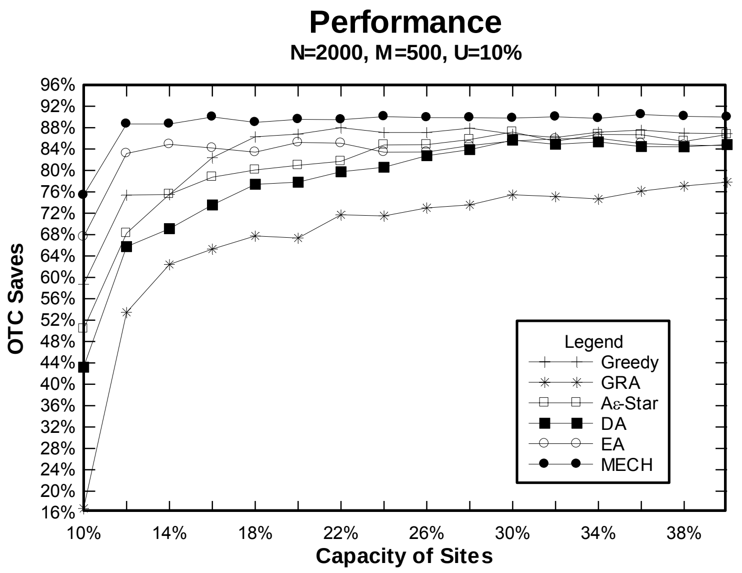

Figure 8.8: OTC savings versus capacity.

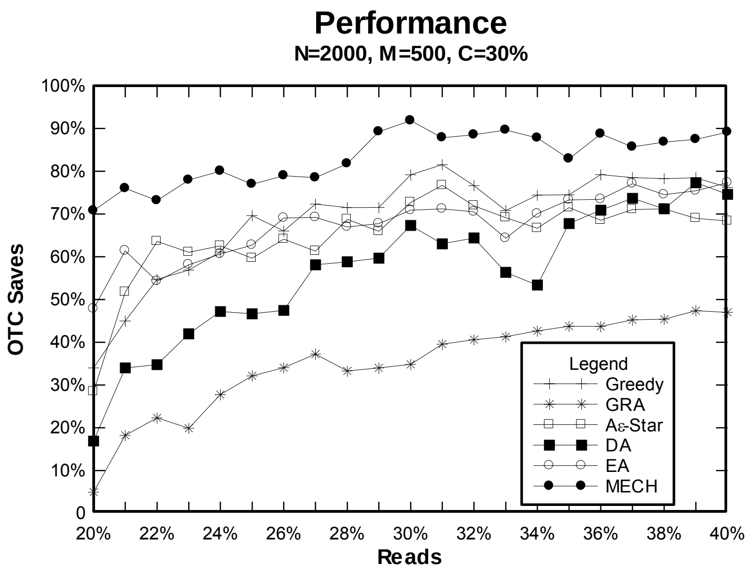

Figure 8.9: OTC savings versus reads.

showed initial loss in OTC savings with the initial number of site increase in the system, as much as 5% loss was recorded in case of MECH with only a 40 site increase. GRA and Greedy showed an initial gain since with the increase in the number of sites, the population permutations increase exponentially, but with the further increase in the

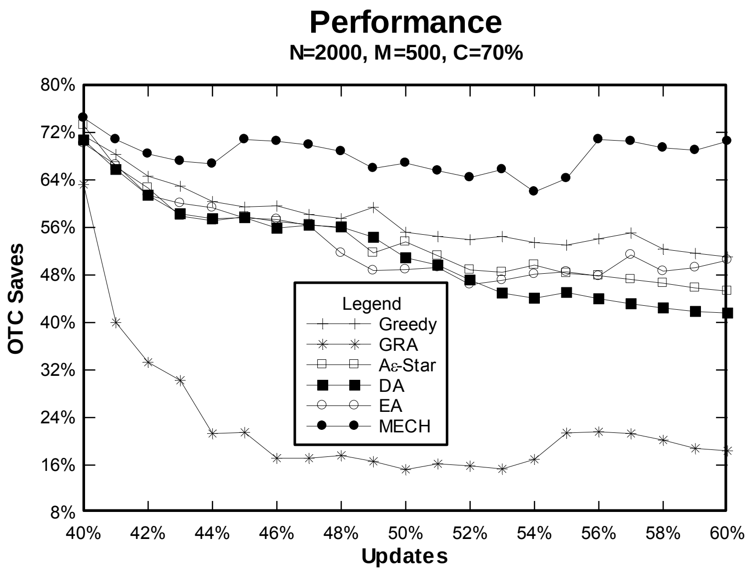

Figure 8.10: OTC savings versus updates.

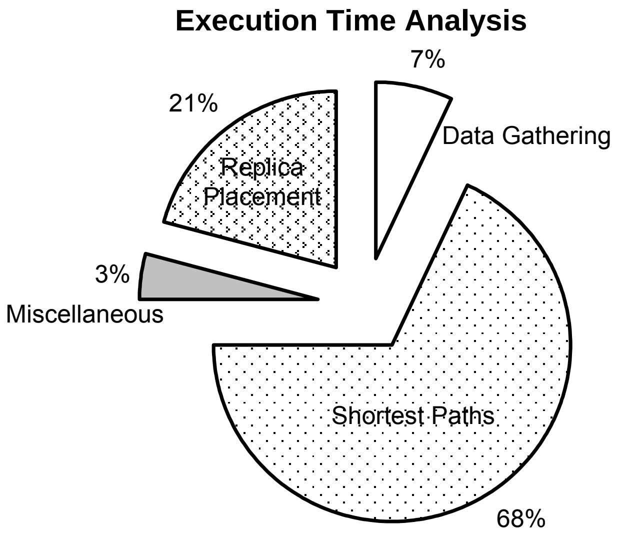

Figure 8.11: Execution time components.

number of sites this phenomenon is not so observable as all the essential objects are already replicated. The top performing techniques (DA, EA, Aε-Star and MECH) showed an almost constant performance increase (after the initial loss in OTC savings). 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 four equally cater for the two diverse effects. GRA also showed a similar trend but maintained lower OTC savings. This was in line with the claims presented in [57] and [78].

To observe the effect of increase in the number of objects in the system, we chose a softer workload with C = 15% and U = 40%. The intention was to observe the trends for all the algorithms under various workloads. 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. From the plot, the most surprising result came from GRA and Greedy. They dropped their savings from 62% to 2% and 69% to 3%, respectively. This was contradictory to what was reported in [78] and [100]. But there the authors had used a uniformly distributed link cost topology, and their traffic was based on the Zipf distribution [123]. While the traffic access logs of the World Cup 1998 are more or less double-Pareto in nature. In either case the exploits and limitations of the technique under discussion are obvious. The plot also shows a near identical performance by Aε-Star, DA and Greedy. The relative difference among the three techniques is less than 3%. However, Aε-Star did maintain its domination. From the plots the supremacy of EA and MECH is observable.

Next, we observe the effects of system capacity increase. An increase in the storage capacity means that a large number of objects can be replicated. Replicating an

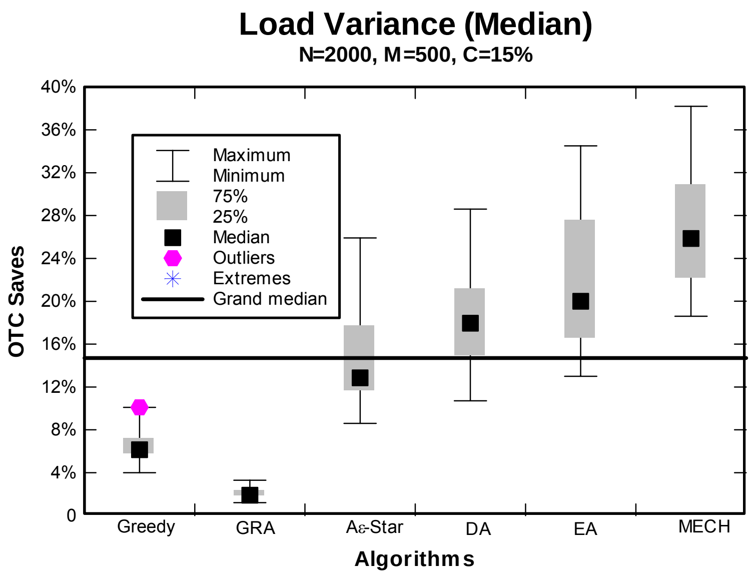

Figure 8.12: Median load variance.

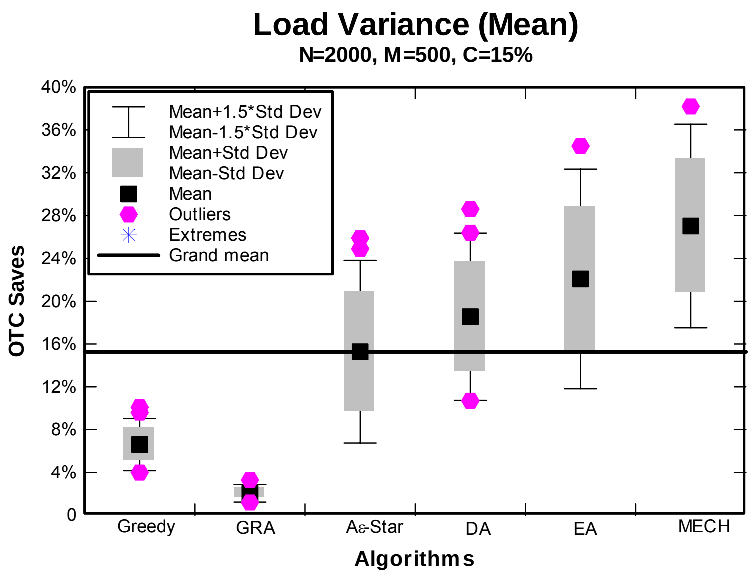

Figure 8.13: Mean load variance.

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. This is observable in Figure 8.8, which shows the performance of the algorithms. GRA once again performed the worst. The gap between all other approaches was reduced to within 15% of each other. DA and MECH showed an immediate initial increase (the point after which further replicating objects is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most OTC savings (53%) followed by Greedy with 34%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in capacity from 13% to 24%, resulted in 4.3 times (on average) more replicas for all the algorithms.

Next, we observe the effects of increase in the read and update (write) frequencies. Since these two parameters are complementary to each other, we 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 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

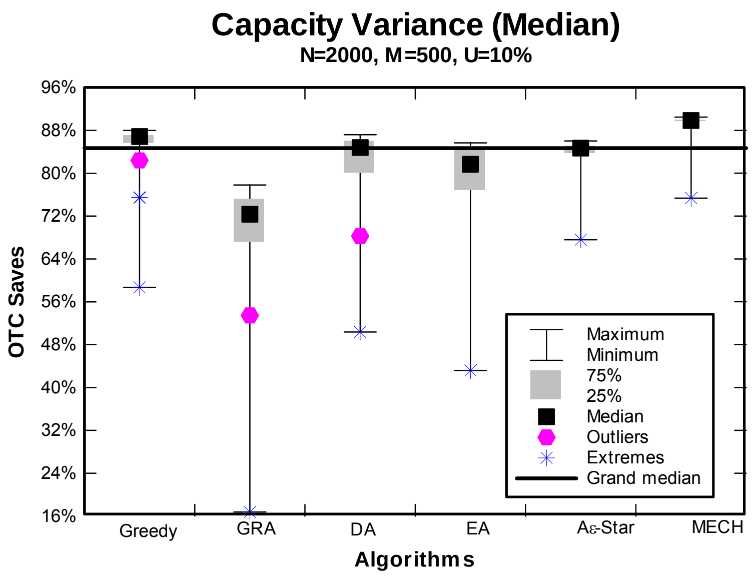

Figure 8.14: Median capacity variance.

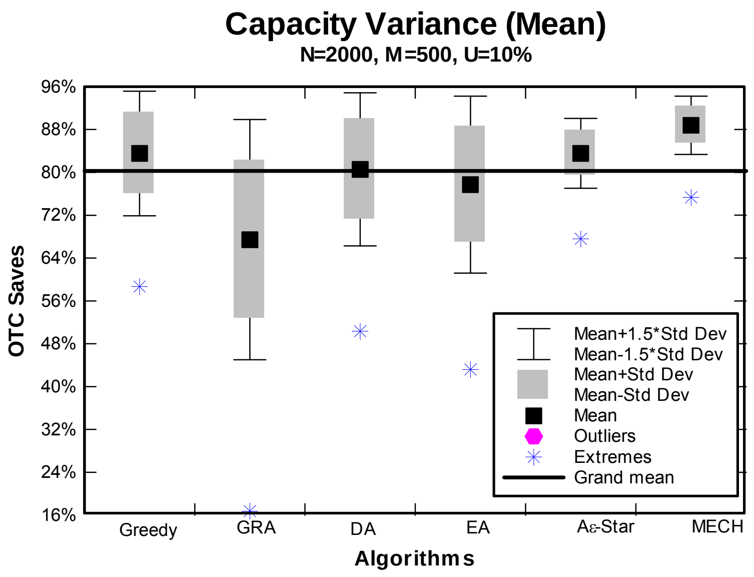

Figure 8.15: Mean capacity variance.

marginal algorithms. The plots in Figures 8.9 and 8.10 show the results of read and update frequencies, respectively. A clear classification can be made between the algorithms. Aε-Star, DA, EA, Greedy and MECH incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 89%. Aε-

Star gained the most of the OTC savings of up to 47%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depend on the initial population (for details see [78]). Moreover, GRA maintains a localized network perception. 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, Aε-Star, DA, EA, Greedy never tend to deviate from their global view of the problem domain.

Lastly, we compare the termination time of the algorithms. Before we proceed, 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 Figure 8.11. There 68% of all the algorithm termination time was taken by the repeated calculations of the shortest paths. Data gathering and dispersion, such as reading the access frequencies from the processed log, etc. took 7% of the total time. Other miscellaneous operations including I/O 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 that 21% of the time depicted in Tables 8.1 and 8.2.

Various problem instances were recorded with C = 20%, 35% and U = 25%, 35%. Each problem instance represents the average recorded time over all the 45 topologies and 13 various access logs. The entries in bold represent the fastest time recorded over the problem instance. It is observable that MECH and DA terminated faster than all the other techniques, followed by EA, Greedy, Aε-Star and GRA. If a static environment was considered, MECH with the maximum problem instance would have terminated approximately in 55.16 seconds (21% of the algorithm termination time).

In summary, based on the solution quality alone, the algorithms can be classified into four categories: 1) Very high performance: EA and MECH, 2) high performance: Greedy and DA, 3) medium-high performance: Aε-Star, and finally 4) mediocre performance: GRA. Considering the execution time, MECH and DA did extremely well, followed by EA, Greedy, Aε-Star, and GRA.

Table 8.3 shows the quality of the solution in terms of OTC percentage for 10 problem instances (randomly chosen), each being a combination of various numbers of sites and objects, with varying storage capacity and update ratio. For each row, the best result is indicated in bold. The proposed MECH algorithm steals the show in the context of solution quality, but Aε-Star, EA and DA do indeed give a good competition, with a savings within 5%-10% of MECH.

8.4.3 Comparative Supergame Analysis

Here, we present some supplementary results regarding the supergame that strengthen our comparative analysis claims provided in Section 8.4.2. We show the relative performance of the techniques with load and storage capacity variance. The plots in Figures 8.12-8.15 show the recorded performances. All the plots summarize the measured performance with varying parameters observed over a time period of 86

Table 8.1: Running time in seconds [*C*=20%, *U*=25%].

Problem Size	Greedy	GRA	Aε-Star	DA	EA	MECH
M=20, *N*=50	69.76	92.57	97.02	**24.66**	39.29	25.24
M=20, *N*=100	76.12	96.31	102.00	26.97	40.91	**26.35**
M=20, *N*=150	78.11	100.59	113.79	**31.98**	53.85	35.64
M=30, *N*=50	94.33	125.93	139.98	38.20	58.98	**38.05**
M=30, *N*=100	108.18	124.20	148.03	**38.29**	62.97	39.60
M=30, *N*=150	134.97	148.49	178.84	44.97	67.74	**42.02**
M=40, *N*=50	126.25	153.93	198.11	**42.34**	75.88	44.66
M=40, *N*=100	134.06	168.09	236.48	**43.54**	76.27	46.31
M=40, *N*=150	140.30	204.12	270.10	**47.02**	82.44	48.41

Table 8.2: Running time in seconds [*C*=35%, *U*=35%].

Problem Size	Greedy	GRA	Aε-Star	DA	EA	MECH
M=300, *N*=1450	206.26	326.82	279.45	**95.64**	178.9	97.98
M=300, *N*=1500	236.61	379.01	310.12	115.19	185.15	**113.65**
M=300, *N*=1550	258.45	409.17	333.03	127.1	191.24	**124.73**
M=300, *N*=2000	275.63	469.38	368.89	143.94	197.93	**142.16**
M=400, *N*=1450	321.6	492.1	353.08	**176.51**	218.15	176.90
M=400, *N*=1500	348.53	536.96	368.03	**187.26**	223.56	195.41
M=400, *N*=1550	366.38	541.12	396.96	**192.41**	221.1	214.55
M=400, *N*=2000	376.85	559.74	412.17	**208.92**	245.47	218.73
M=500, *N*=1450	391.55	659.39	447.97	**224.18**	274.24	235.17
M=500, *N*=1500	402.2	660.86	460.44	**246.43**	284.63	259.56
M=500, *N*=1550	478.1	689.44	511.69	**257.96**	301.72	266.42
M=500, *N*=2000	485.34	705.07	582.71	269.45	315.13	**262.68**

Table 8.3: Average OTC (%) savings under some problem instances.

Problem Size	Greedy	GRA	Aε-Star	DA	EA	MECH
N=150, *M*=20 [*C*=20%,*U*=25%]	70.27	69.11	73.96	69.91	72.72	**74.40**
N=200, *M*=50 [*C*=20%,*U*=20%]	73.49	69.33	76.63	71.90	**77.11**	75.43
N=300, *M*=50 [*C*=25%,*U*=5%]	69.63	63.45	69.85	67.66	69.80	**70.36**
N=300, *M*=60 [*C*=35%,*U*=5%]	71.15	64.95	71.51	69.26	70.38	**74.03**
N=400, *M*=100 [*C*=25%,*U*=25%]	67.24	61.74	71.26	68.67	70.49	**73.26**
N=500, *M*=100 [*C*=30%,*U*=35%]	65.24	60.77	70.55	69.82	70.87	**72.73**
N=800, *M*=200 [*C*=25%,*U*=15%]	66.53	65.90	69.33	68.95	70.06	**72.95**
N=1000, *M*=300 [*C*=25%,*U*=35%]	69.04	63.17	69.98	69.36	71.28	**72.44**
N=1500, *M*=400 [*C*=35%,*U*=50%]	69.98	62.61	70.41	72.09	72.26	**72.78**
N=2000, *M*=500 [*C*=10%,*U*=60%]	66.34	62.70	71.33	67.67	68.41	**74.06**

simulation days (this is the entire time period of the logs that are available for the World Cup 1998 web server). Notice that the supergame setup is tested over all the available access logs. We are mostly interested in measuring the median and mean performances of the algorithms. With load variance MECH edges over EA with a savings of 39%. The plot also shows that nearly every algorithm performed well with a grand median of

15.9%. The graphs are self explanatory in nature, and also capture the outliners and extreme points. The basic exercise in plotting these results is to see which algorithms perform consistently over an extended period of time. GRA for example, records the lowest extremes, and hardly any outliners. On the other hand the proposed MECH's performance is captured in a small interval, with high median and mean OTC savings. The readers may notice the difference in the performance of the algorithms with load and capacity variances. This is because load variance captures all the possible combinations of read and update parameters. For example, in a network with 100% updates there will hardly be any measurable OTC Savings. Thus, Figures 8.12 and 8.13 show mediocre OTC savings, simply because they encapsulated the performance of the networks where update ratio was extremely high.

8.5 Concluding Remarks

A game theoretical discriminatory mechanism (MECH) for fine-grained data replication in large-scale distributed computing systems (e.g. the Internet) was proposed. In MECH we employ agents who represent data objects to compete for the limited available storage space on web servers to acquire the rights to replicate. MECH uses a unique concept of supergame in which these agents continuously compete in a non-cooperative environment. MECH allows the designers the flexibility to monitor the behavior and strategies of these agents and fine-tune them so as to attain a given objective. In case of the data replication problem, the object for these agents is to skillfully replicate data objects so that the total object transfer cost is minimized.

MECH was compared against some well-known techniques, such as: greedy, branch and bound, game theoretical auctions and genetic algorithms. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The experimental results revealed that MECH outperformed the five widely cited and powerful techniques in both the execution time and solution quality.

In summary, MECH exhibited 5%-10% better solution quality and 25%-35% savings in the algorithm termination timings.

CHAPTER 9

A BUDGET BALANCED GAME THEORETICAL REPLICA PLACEMENT TECHNIQUE

We introduce an agent-based distributed budget balanced game theoretical replica placement, allocation, and management technique, where each agent maximizes its own benefit, such as, user access time, latency and communication cost. The proposed technique gathers inspiration from market economy and game theoretical mechanism designs. In such mechanisms the agents do not have a global view of the system, which makes the optimization process highly localized. This local optimization may encourage these agents to alter the output of the resource allocation mechanism in their favor and act selfishly. The proposed technique guarantees a global optimal solution even though the system acts in a distributed fashion operated by self-motivated selfish agents.

9.1 Introductory Views

We propose a simple approach to designing resource allocation mechanisms for autonomous distributed computing systems. The approach draws inspiration from game theory and the similarities between market economics and large-scale distributed computing systems.

Just like in a market economy, a large-scale distributed computing system has scarce (computational) resources such as: processing power, memory, network bandwidth, etc. In market economy resources are managed by decentralized autonomous agents. We seek to exploit the lessons learnt from the evolved market economy and effectively apply them to replicate and manage data objects in a large-scale distributed computing system such as the Internet.

Replicating the data over geographically dispersed locations reduces access latency, network traffic, and in turn adds reliability, robustness and fault-tolerance to the system. Discussions in [46], [57], [75], [78] and [100] reveal that client(s) experience reduced access latencies provided that data is replicated within their close proximity. However, this is applicable in cases when only read accesses are considered. If updates of the contents are also under focus, then 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. Therefore, efficient and effective replication schemas strongly depend on how many replicas to be placed in the system, and more importantly where.

In our game theoretical replica allocation and management mechanism (RAMM), each site (node) is represented by an agent. We view an agent as part of a community of similar though heterogeneous agents that are designed to compete for scarce resources. Motivated by their self interests and the fact that the agents do not have a global view of the distributed system, they optimize their individual interests, such as, minimize communication costs, latencies, etc. Each agent defines its goals and utilities, and the rules for max(min)imization. Although no direct attempt is made to globally improve or optimize the system wide goals, yet the mechanism provides a platform for self-evolving solution quality. This results in global performance improvement through an invisible hand.

We evaluate our proposed approach through a simulation study of a large-scale distributed computing system mimicking the Internet, and compare it with five various techniques recorded in the literature. Experimental results reveal that our proposed approach improves performance relative to these techniques in three ways. First, the number of replicas in a system is controlled to reflect the ratio of read versus write access. To maintain concurrency control, when an object is updated, all of its replicas need to be updated simultaneously. If the write access rate is high, there should be few replicas to reduce the update overhead. If the read access rate is overwhelming, there should be a high number of replicas to satisfy local accesses. Second, performance is improved by replicating objects to the sites based on locality of reference. This increase the probability that object access can be satisfied either locally or within a desirable amount of time from a neighboring site. Third, replica assignments are made in a fast algorithmic turn-around time. All the above improvements are achieved by a simple, decentralized, and autonomous RAMM.

In addition to the performance improvements above, RAMM offers other benefits. The most important of them all is that the complexity is decreased by multifold. RAMM limits the complexity by partitioning the complex global problem of replica allocation, into a set of simple independent sub problems. Each agent independently attempts to optimize its utility. RAMM also unifies the selfish optimization of the participating agents into a globally effective replica allocation. This approach is well suited to the large-scale distributed computing systems that are composed of autonomous agents which do not necessarily cooperate to improve the system wide goals, but provide a framework for self-stabilization and repair.

9.2 The Replica Allocation and Management Mechanism (RAMM)

According to the definition in [66], an auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants. This definition allows us to formulate a generalized mechanism as:

1. Agents send bids to the mechanism to indicate their willingness to exchange goods.
2. The auction may post price quotes to provide summarized information about the status of the price-determination process.

   Steps 1 and 2 may be iterated.
3. The auction determines an allocation and notifies the agents as to who purchases what from whom at what price.

The above sequence may be performed once or repeated any number of times. In this paper, the mechanism we discuss is decentralized in the sense that each agent calculates its own bidding strategy, based on *local information*.

Unlike the more popular types of auctions such as the English and the Dutch auctions, the Generalized Vickrey Auction (GVA) is a direct revelation mechanism

([24], [40], [114]), and thus is not a price system. Rather, it computes overall payments for agent's allocations that sometimes, but not always, translate into meaningful prices. If agents play Bayesian-Nash or dominant strategies, any desirable choice function that can be implemented by a mechanism is quite powerful. Specifically, the GVA is a direct revelation mechanism on dominant strategies in the class of Groves [40] and Clark [24] mechanisms. In [38] authors have shown under rather general conditions that when agents have quasi-linear preferences, the only efficient social choice functions that are implemented in dominant strategies are those that are implemented by Groves-Clarke mechanism.

From above we can conclude that an efficient, optimal and computationally feasible mechanism should possess the following properties:

1. Agent's have quasi-linear preferences.
2. Agent's have dominating strategies.

Under our (data replication) problem formulation, if we can prove the (above) two properties, than the Groves-Clarke mechanism would be sufficient. In the subsequent text we shall do exactly the same.

9.2.1 Preliminaries

9.2.1.1 The Basics

The mechanism contains *M* agents. Each agent *i* has some private data $t^i \in \mathbf{R}$. This data is termed as the agent's *true data* or *true type*. Only agent *i* has knowledge of

t^i. Everything else in the mechanism is public knowledge. Let t denote the vector of all the true types $t = (t^1 \ldots t^M)$.

9.2.1.2 Communications

The only information that is relayed to the mechanism by an agent i is its corresponding bid b^i. Since the agents are selfish in nature, (*i.e.*, localized optimization) they *may* ($b^i = t^i$) or *may not* ($b^i \neq t^i$) communicate to the mechanism the value t^i. Let b denote the vector of all the bids (($b = (b^1 \ldots b^M)$)), and let b^{-i} denote the vector of bids, not including agent i, *i.e.*, $b^{-i} = (b^1 \ldots b^{i-1}, b^{i+1}, \ldots b^M)$. It is to be understood that we can also write $b = (b^{-i}, b^i)$.

9.2.1.3 Components

The mechanism has two components: 1) the algorithmic output $x(\cdot)$, and 2) the payment mapping function $p(\cdot)$.

9.2.1.4 Algorithmic Output

The mechanism allows a set of outputs X, based on the output function which takes in as the argument, the bidding vector, *i.e.*, $x(b) = \{x^1(b), \ldots, x^M(b)\}$, where $x(b) \in X$. This output function relays a unique output given a vector b. That is, when $x(\cdot)$ receives b, it generates an output which is of the form of allocations $x^i(b)$. Intuitively it would mean that the algorithm takes in the vector bid b and then relays to each agent its allocation.

9.2.1.5 Monetary Cost

Each agent i incurs some monetary cost $c^i(t^i,x^i(b))$, *i.e.*, the cost to accommodate the (data) allocation $x^i(b)$. This cost is dependent upon the output (of the allocations by the mechanism $x^i(b)$) and the agent's private data t^i.

9.2.16 Payments

To offset c^i, the mechanism makes a payment $p^i(b)$ to agent i. An agent i always attempts to maximize its profit (utility) $u^i(t^i,b) = p^i(b) - c^i(t^i,x^i(b))$. Each agent i cares about the other agents' bid only insofar as they influence the outcome and the payment.

9.2.1.7 Bids

Each agent i is interested in reporting a bid b^i such that it maximizes its profit, regardless of what the other agents bid, *i.e.*, $u^i(t^i,(b^{-i},t^i)) \geq u^i(t^i,(b^{-i},b^i))$ for all b^{-i} and b^i. It is to be noted that truth telling ($b^i = t^i$) brings in more utility to the agents because the following do not hold:

1. *Over projection:* Agents in anticipation of more revenue over project their true data, but this does not help, as the agent who is allocated the object gets the second best payment. Note that in Groves-Clarke mechanism second best payment is a strong tool to confine the agents from misreporting.
2. *Under projection:* If every agent under projects their true data, that does not help either as the revenue would drop in proportion to the under projection.
3. *Random projection:* In this case the deserving agent would be at loss. Therefore, it

is unlikely that a selfish agent would agree to project random true data.

4. For more details on the optimality of such type of payment procedure see [106]. In that paper, the authors have identified many such scenarios, all but reporting truthfully fail to exploit this (second best) payment option.

9.2.2 The RAMM

We now put all the pieces together. A mechanism *m* consists of a pair $m = (x(b),p(b))$, where $x(\cdot)$ is the output function and $p(\cdot)$ is the payment mapping function. The objective of the mechanism is to select an output *x*, that optimizes a given objective function.

9.2.2.1 Objective

The mechanism defined above leaves us with the following two optimization problems:

1. Identify a strategy that is dominant to each agent *i*.
2. Identify a payment mapping function that is truthful.

9.2.2.2 The Basic Results

From previous discussion recall that we carry with us the following three pending questions:

1. Agent's preferences should be quasi-linear.
2. Agent's strategies should depict dominance.

3. Payments should implement truthfulness. (We answer them below.)

9.2.2.2.1 Quasi-linear Preferences

Quasi-linearity implies that the mechanism is able to make any cash transfer that exactly compensates any agent for any possible change in outcomes, and that redistributing wealth among the agents would not change this compensatory transfer. In such a setup any agent's payoff (utility) is given by: $u^i(t^i,b) = p^i(b) - c^i(t^i,x^i(b))$ [40]. This payoff implies that each agent cares about his own cash (received) payment (from the mechanism), but not about payments that other agents receive. This is exactly what the RAMM's payment functions do.

9.2.2.2.2 Dominating Strategy

The agents in the mechanism value an object *k* for the benefit that it brings to the agent's site *i*. This benefit is equivalent to the savings that the object *k* brings in the total object transfer cost (OTC) if the object *k* is replicated at site *i*. This benefit is:

$$B_k^i = RC_k^i - \sum_{x=1}^{M} w_k^x o_k c(i,P_k) .$$

We discussed the optimality of the above stated benefit cost function in Chapter 8. We strongly suggest readers to review before proceeding any further.

9.2.2.2.3 Payments

The mechanism eliminates incentives for misreporting by imposing on each agent the cost of any distortion it causes. The payment for agent *i* is set so that *i*'s report cannot effect the total payoff to the set of other agents (excluding agent i), M-i.

To capture the effect of i's report on the outcome, we introduce a hypothetical *null report*, which corresponds to agent i reporting that it is indifferent among the possible decisions and cares only about payments. When i makes the null report, the mechanism optimally chooses the decision $D(X,M\text{-}i,t^{-i})$. The resulting total value of the decision for the set of agents M-i would be $V(X,M\text{-}i,t^{-i})$, and the mechanism might also provide an agent i with payment equivalent to $h^i(t^{-i})$. Thus, if i makes a null report, the total payoff to the agents in set M-i is $V(X,M\text{-}i,t^{-i}) + h^i(t^{-i})$. This would mean that the RAMM would choose payments for the M-i agents regardless of what i reports to the RAMM. For a detailed analysis of the above payment structure, readers are encouraged to see [58] and [61]. It is to be noted that in economic game theoretical literature this type of payment is often referred to as Vickrey payments [94].

We have entertained all the pending optimization issues regarding the RAMM, and are ready to give a pseudo-code (Figure 9.1).

Briefly, we maintain a list L^i at each server. This list contains all the objects that can be replicated by agent i onto site S^i. We can obtain this list by examining the two constraints of the DRP. List L^i would contain all the objects that have their size less then the total available space b^i. Moreover, if site S^i is the primary host of some object k', then k' should not be in L^i. We also maintain a list LS containing all sites that can replicate an object, *i.e.*, $S^i \in LS$ if $L^i \neq$NULL. The algorithm works iteratively. In each step the mechanism asks all the agents to send their preferences (first **PARFOR** loop). Each agent i recursively calculates the true data of every object in list L^i. Each agent

The RAMM Algorithm

```
Initialize:
$LS, L^i, T_k^i, M, MT$

01 WHILE $LS \neq$ NULL DO
02   $OMAX$ = NULL; $MT$ = NULL; $P^i$ = NULL;
03      PARFOR each $S^i \in$LS DO
04            FOR each $O_k \in L^i$ DO
05                 $T_k^i$ = compute ($B_k^i$);  /*compute the valuations/bids*/
06            ENDFOR
07            $t^i$ = argmax$_k$($T_k^i$);
08            SEND $t^i$ to M; RECEIVE at $M$ $t^i$ in $MT$;
09       ENDPARFOR
10  OMAX = argmax$_k$($MT$);  /*Choose the global dominate valuation/bid*/
11  DELETE $k$ from $MT$;
12  $P^i$ = argmax$_k$($MT$);             /*Calculate the payment*/
13  BROADCAST $OMAX$;
14  SEND $P^i$ to $S^i$; RECEIVE at $S^i$ /*Pay the winning agent this amount*/
15  Replicate $O_{OMAX}$;
16  $ac^i$=$ac^i$ - $o_k$;                    /*Update capacity*/
17  $L^i$ = $L^i$ - $O_k$;                 /*Update the list*/
18 IF $L^i$ = NULL THEN SEND info to $M$ to update $LS = LS - S^i$;       /*Update mechanism players*/
19      PARFOR each $S^i \in$LS DO
20           Update $NN^i_{OMAX}$          /*Update the nearest neighbor list*/
21      ENDPARFOR                /*Get ready for the next round*/
22 ENDWHILE
```

Figure 9.1: Pseudo-code describing the RAMM.

then reports the dominant true data (line 08) to the mechanism. The mechanism receives all the corresponding entries, and then chooses the best dominant true data. This is broadcasted to all the agents, so that they can update their nearest neighbor table NN_k^i, which is shown in Line 20 (NN^i_{OMAX}). The object is replicated and payments made to the agent. The mechanism progresses forward till there are no more agents interested in acquiring any data for replication (Line 18).

The above discussion allows us to deduce the following result about the RAMM algorithm.

Theorem 9.1: *In the worst case the RAMM takes $O(MN^2)$ time.*

Proof: The worst case scenario is when each site has sufficient capacity to store all objects. In that case, the while loop (Line 02) performs *MN* iterations. The time complexity for each iteration is governed by the two **PARFOR** loops (Lines 04 and 19). The first loop uses at most *N* iterations, while the send loop performs the update in constant time. Hence, we conclude that the worst case running time of the mechanism is $O(MN^2)$. ■

9.3 Experimental Comparative Analysis

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory using the experimental infrastructure as described in Chapter 3. The experimental evaluations were targeted to benchmark the placement policies. The RAMM was implemented using IBM Pthreads.

9.3.1 Comparative Algorithms

For comparison, we selected five various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The techniques studied include efficient branch-and-bound based technique (Aε-Star [57]). For fine-grained replication, the algorithms proposed in [75], [78], and [100] are the only ones that address the problem domain similar to ours. We select from [100] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [75]); thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [58] (Dutch (DA) and English auctions (EA)) and [78] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons.

9.3.2 Performance Metric

The solution quality is measured in terms of network communication cost (OTC percentage) that is saved under the replication scheme found by the algorithms, compared to the initial one, *i.e.*, when only primary copies exists.

9.3.3 Comparative Analysis

We study the behavior of the placement techniques when the number of sites increases (Figure 9.2), by setting the number of objects to 2000, while in Figure 9.3, we study the behavior when the number of objects increase, by setting the number of sites to 500. We should note here that the space limitations restricted us to include various other scenarios with varying capacity and update ratio. The plot trends were similar to the ones reported in this article. For the first experiment we fixed *C*=35% and *U*=70%. We intentionally chose a high workload so as to see if the techniques studied successfully handled the extreme cases. The first observation is that RAMM and EA outperformed other techniques by considerable amounts. Second, DA converged to a better solution quality under certain problem instances. Some interesting observations were also recorded, such as, all but GRA and Greedy showed initial loss in OTC savings with the initial number of site increase in the system, as much as 5% loss was recorded in case of Aε-Star with only a 40 site increase. GRA and Greedy showed an

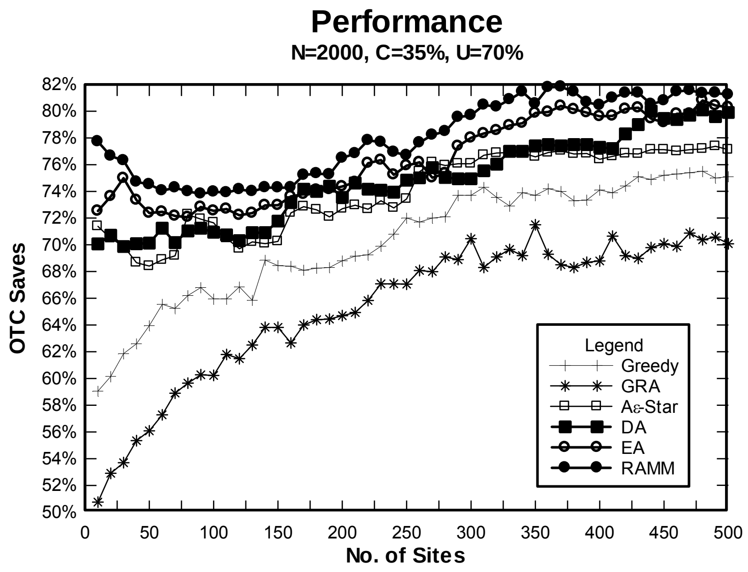

Figure 9.2: OTC savings versus number of sites.

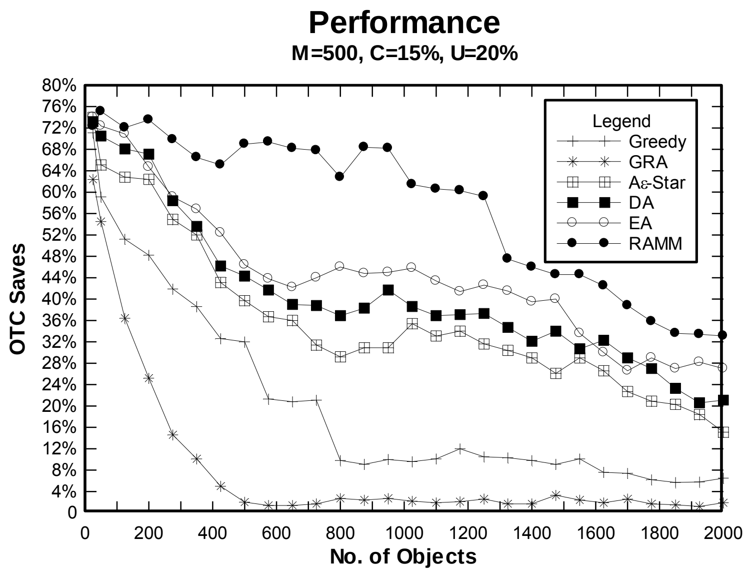

Figure 9.3: OTC savings versus number of objects.

initial gain since with the increase in the number of sites, the population permutations increase exponentially, but with the further increase in the number of sites this phenomenon is not so observable as all the essential objects are already replicated. The top performing techniques (DA, EA, Aε-Star and RAMM) showed an almost constant

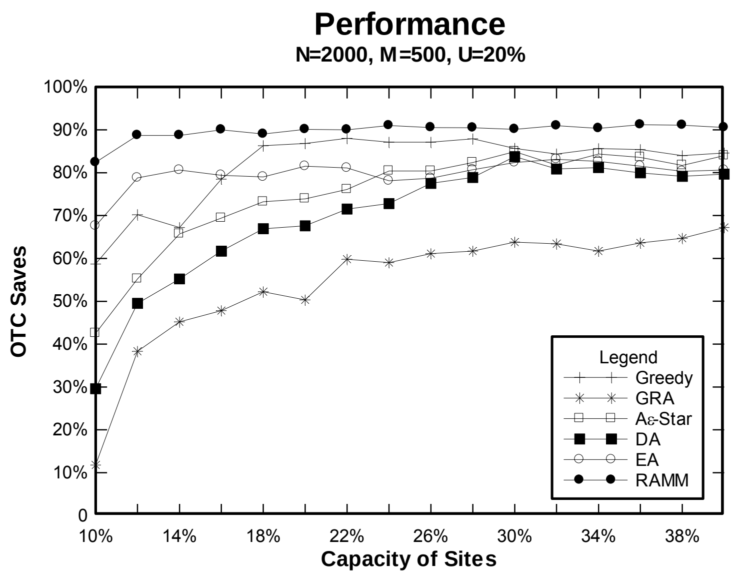

Figure 9.4: OTC savings versus capacity.

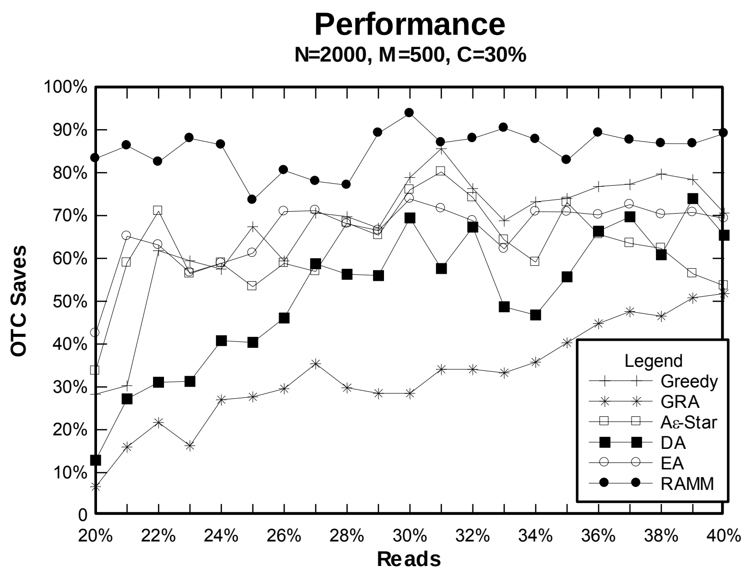

Figure 9.5: OTC savings versus reads.

performance increase (after the initial loss in OTC savings). 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 four equally cater for the two diverse effects. GRA and Greedy also showed a similar trend but maintained lower OTC savings. This was in line with the claims presented in [57] and [78].

To observe the effect of increase in the number of objects in the system, we chose a softer workload with C = 15% and U = 20%. The intention was to observe the trends for all the algorithms under various workloads. 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. From the plot, the most surprising result came from GRA. It dropped its savings from 63% to 2%. This was contradictory to what Was reported in [78]. But there the authors had used a uniformly distributed link cost topology, and their traffic was based on the Zipf distribution [123]. While the traffic access logs of the World Cup 1998 are more or less double-Pareto in nature. In either case the exploits and limitations of the technique under discussion are obvious. The plot also shows a near identical performance by Aε-Star, DA and EA. The relative difference among the three techniques is less than 4%. However, EA did maintain its domination. From the plot (Figure 6) the supremacy of EA and RAMM is observable.

Next, we observe the effects of system capacity increase. 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. This is observable in Figure 9.4, which shows the performance of the algorithms. GRA once again performed the worst. The gap between all other approaches was reduced to within 12% of each other. DA and RAMM showed an immediate initial increase (the point after which further replicating objects is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most OTC savings (47%) followed by Greedy with 44%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in capacity from 10% to 17%, resulted in 3.7 times (on average) more replicas for all the algorithms.

Next, we observe the effects of increase in the read and update (write) frequencies. Since these two parameters are complementary to each other, we 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 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.

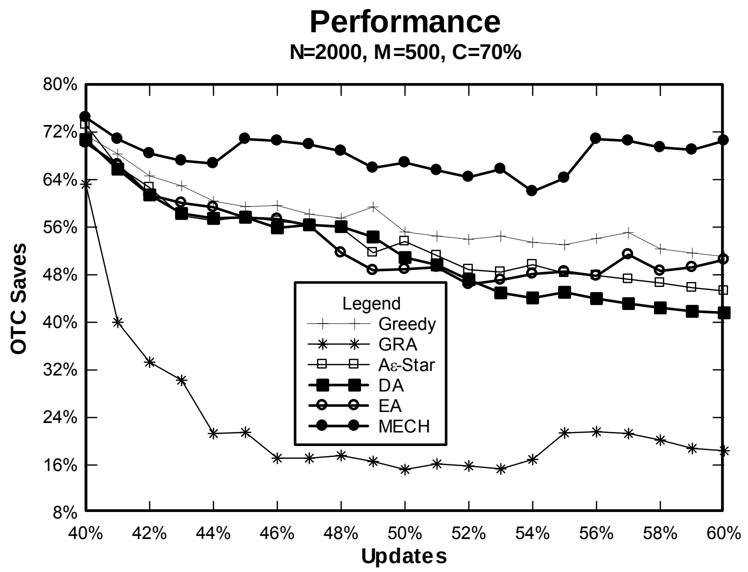

Figure 9.6: OTC savings versus updates.

Table 9.1: Running time (sec.) [*C*=55%, *U*=10%].

Problem Size	Greedy	GRA	Aε-Star	DA	EA	RAMM
M=300, *N*=1400	206.26	326.82	279.45	**95.64**	178.9	97.98
M=300, *N*=1450	236.61	379.01	310.12	115.19	185.15	**113.65**
M=300, *N*=1500	258.45	409.17	333.03	127.1	191.24	**124.73**
M=300, *N*=1550	275.63	469.38	368.89	**143.94**	197.93	147.16
M=300, *N*=2000	298.12	475.02	387.94	**158.45**	204.29	159.12
M=400, *N*=1400	348.53	536.96	368.03	**187.26**	223.56	195.41
M=400, *N*=1450	366.38	541.12	396.96	**192.41**	221.1	214.55
M=400, *N*=1500	376.85	559.74	412.17	**208.92**	245.47	218.73
M=400, *N*=1550	389.71	605.63	415.55	**215.24**	269.31	223.92
M=400, *N*=2000	391.55	659.39	447.97	**224.18**	274.24	235.17
M=500, *N*=1400	478.1	689.44	511.69	**257.96**	301.72	266.42
M=500, *N*=1450	485.34	705.07	582.71	**269.45**	315.13	272.68
M=500, *N*=1500	511.06	736.43	628.23	**278.15**	324.26	291.83
M=500, *N*=1550	525.33	753.5	645.26	**289.64**	331.57	304.47
M=500, *N*=2000	539.15	776.99	735.36	**312.68**	345.94	317.60

The read and update parameters indeed help in drawing a line between good and marginal algorithms. The plots in Figures 9.5 and 9.6 show the results of read and update frequencies, respectively. A clear classification can be made between the algorithms. Aε-Star, EA, Greedy and RAMM incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 86%. GRA gained the least of the OTC savings of up to 13%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial population (for details see [78]). Moreover, GRA maintains a localized network perception. 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, Aε-Star, EA, Greedy and RAMM never tend to deviate from their global view of the problem search space.

Lastly, we compare the termination time of the algorithms. Before we proceed, 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 is as follows. 68% of all the algorithm termination time was taken by the repeated calculations of the shortest paths. Data gathering and dispersion, such as reading the access frequencies from the processed log, etc. took 7% of the total time. Other miscellaneous operations including I/O were recorded to carry 3% of the total execution time. Therefore, a totally static setup would take no less that (100-(68+7+3)) = 21% of the time depicted in Tables 9.1 and 9.2. Various problem instances were recorded with C = 15%, 55% and U = 10%, 55%. The entries in bold represent the fastest time recorded over the problem instance. It is observable that RAMM and DA terminated faster than all the other techniques, followed by EA, Greedy, Aε-Star and GRA. If a static environment was considered, RAMM with the maximum problem instance would have terminated in 66.69 seconds

Table 9.2: Running time (sec.) [*C*=15%, *U*=55%].

Problem Size	Greedy	GRA	Aε-Star	DA	EA	RAMM
M=20, *N*=50	70.06	92.35	96.31	**24.35**	38.69	26.06
M=20, *N*=100	76.20	96.31	102.81	**26.97**	40.39	**26.97**
M=20, *N*=150	77.55	100.93	113.25	**31.62**	53.69	35.98
M=30, *N*=50	95.00	126.80	140.69	**38.31**	59.20	38.85
M=30, *N*=100	108.79	124.55	148.07	**39.01**	62.73	39.40
M=30, *N*=150	135.09	147.67	179.27	45.22	67.91	**41.21**
M=40, *N*=50	125.55	154.11	198.21	**41.79**	76.20	45.11
M=40, *N*=100	134.03	167.56	235.97	**43.25**	77.16	46.19
M=40, *N*=150	140.81	203.54	269.88	**46.91**	81.70	48.39

Table 9.3: Average OTC (%) savings.

Problem Size	Greedy	GRA	Aε-Star	DA	EA	RAMM
N=200, *M*=50 [*C*=20%,*U*=20%]	73.50	70.02	76.45	71.70	**76.50**	75.47
N=300, *M*=50 [*C*=25%,*U*=5%]	69.16	64.17	70.04	67.72	70.02	**70.39**
N=400, *M*=100 [*C*=25%,*U*=25%]	66.52	61.51	70.76	68.63	69.96	**73.19**
N=500, *M*=100 [*C*=30%,*U*=35%]	65.89	61.20	70.71	70.11	70.95	**72.92**
N=800, *M*=200 [*C*=25%,*U*=15%]	66.72	65.57	69.98	68.46	69.83	**72.30**
N=1000, *M*=300 [*C*=25%,*U*=35%]	68.40	63.73	69.89	69.80	70.52	**72.87**
N=1500, *M*=400 [*C*=35%,*U*=50%]	69.79	63.21	69.76	72.23	72.36	**73.14**
N=2000, *M*=500 [*C*=10%,*U*=60%]	66.14	62.89	72.14	68.03	68.29	**73.63**

(approximately 21% of the algorithm termination time (Table 9.1 last entry)).

In summary, based on the solution quality alone, the algorithms can be classified into four categories: 1) The very high performance algorithms that include RAMM and EA, 2) the high performance algorithms of Greedy and DA, 3) the medium-high performance Aε-Star, and finally 4) the mediocre performance algorithm of GRA. While considering the termination timings, RAMM and DA did extremely well, followed by EA, Greedy, Aε-Star, and GRA.

Table 9.3 shows the quality of the solution in terms of OTC percentage for eight problem instances (randomly chosen), each being a combination of various numbers of sites and objects, with varying storage capacity and update ratio. For each row, the best result is indicated in bold. The proposed RAMM algorithm steals the show in the context of solution quality, but Aε-Star, EA and DA do indeed give a good competition, with a savings within a range of 5%-10% of RAMM.

9.4 Concluding Remarks

Manual mirroring of data objects is a tedious and time consuming operation. This study proposed a game theoretical replica allocation and management mechanism (RAMM) for fine-grained data replication in large-scale distributed computing systems such as the Internet. RAMM is a protocol for automatic replication and migration of objects in response to demand changes. RAMM aims to place objects in the proximity of a majority of requests while ensuring that no hosts become overloaded.

RAMM allows agents to compete for the scarce memory space at sites so that they can acquire the rights to place replicas. To cater for the possibility of cartel type behavior of the agents, RAMM uses Vickrey price protocol. This leaves the agents with no option, then to report truthful valuations of the objects that they represent.

RAMM was compared against some well-known techniques, such as: branch and bound, greedy, game theoretical auctions, and genetic algorithms. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The experimental setup was designed to mimic a large-scale distributed computing system (the Internet), by using several Internet topology generators and World Cup Soccer 1998 web server access logs. The experimental results revealed that RAMM outperformed the three widely cited and powerful techniques in both the execution time and solution quality. In summary, RAMM exhibited 5%-10% better solution quality and 10%-30% savings in the algorithm termination timings.

CHAPTER 10

A COOPERATIVE REPLICA PLACEMENT TECHNIQUE

10.1 Introductory Views

A number of replica placement techniques for large distributed computing systems have been proposed with the underlying assumption that the servers cooperate with one another in order to layout a replica schema that optimizes the overall system performance. For instance, almost all content distribution networks (CDNs) related replica placement techniques (e.g. [18], [41], [46], [100]) rely on a centralized decision making body which optimizes a given objective (e.g. to reduce the communication cost) regardless of the costs incurred by each server. These previously reported techniques are plausible as they advance the study of replica placements, however, they are very tedious and have very high computational complexity [21]. For instance, some techniques require that the underlying infrastructure be a tree [46], and the best possible bound (reported in [75]) is of the order of $O(M^3N^2)$, where M is the number of servers and N is the number of (data) objects, respectively, in the system.

To study the cooperative behavior of the servers and to derive a scalable replica placement technique, we make use of game theoretical techniques. Each server in the system plays a cooperative replica placement game (COOP). In COOP each server has two possible actions for each object. If an access is made to an object that is located at a nearby server, then the server is better off redirecting the request to that server. On the other hand if the object is located at a far off server, then the server is better off replicating that object. These decisions by the servers are not taken individually but collectively. The goal of this chapter is to see whether these servers in COOP, can layout replica schemas that converge to global optimum solution(s) targeted towards reducing the communication cost induced by accessing the objects. A cooperative game is defined as a game in which players can conclude a binding agreement as to what outcome will be chosen to exploit the possibility of common interests. Cooperation in the sense of game theory does not mean that players sacrifice their interests for the sake of others, only that each communicates and coordinates its actions for the purpose of furthering their interests. Due to the fact that servers in a large distributed computing system can share resources, all of them should cooperate to obtain the best possible benefit. In this chapter, the Aumann-Shapley resource allocation mechanism of cooperative game theory will be used for the replica placement problem. The proposed methodology not only ensures that the total communication cost is globally minimized, but also that the data allocation is fair leading to load balancing.

10.2 Cooperative Game Theoretical Replica Placement Game

10.2.1 The Aumann-Shapley Mechanism

A natural framework for the study of resource allocation problems is game theory. A game theoretical framework takes into account the strategic aspects of the situation and yields a reasonable concept of unique equilibrium (solution) characterized by the fairness of the allocations.

In game theory, resource allocation problems can be stated as allocating the jointly used resources (in our case the allocation of replicas) among participants in a cooperative game. From game theory point of view, there is only one plausible resource allocation mechanism that is fully distributive and satisfies the fairness principle in sharing as a cooperative game, namely, the Aumann-Shapely mechanism:

$$\psi_i\left(f, x`\right) = \int_0^1 \frac{\partial f(tx`)}{\partial x_i} dt, \tag{10.1}$$

where $\psi_i(f,x`)$ is defined for all possible allocations $(f,x`)$ on some fixed set of inputs, such that $\psi_i(f,x`)$ is the allocation associated with $i$. It is assumed that $f$ has continuous first partial derivatives on some bounded domain of the form $D = D(x`) = \{x \in \Re^n : 0 \leq x \leq x`\}$.

Game theoreticians have proven that the Aumann-Shapley mechanism generates a unique allocation that is continuous, aggregate invariant, fully distributive, and satisfies the fairness principle. We will see that its application to the data replication problem will not generate a mismatch of resource allocation which is the one of the primary reasons for obtaining sub-optimal solutions.

10.2.2 Replica Placement Game (COOP)

As mentioned before, a cooperative game is a game in which the players can conclude a binding agreement as to what outcome will be chosen to exploit the possibility of common interests. In game theory a resource allocation game can be state as dividing the cost of jointly used resources among participants in a cooperative game. Since reducing the overall communication cost is a resource (replica) allocation game, it is appropriate to define the optimization of the communication cost as a cooperative game.

10.2.3 Aumann-Shapley Replica Placement Game

Suppose there are a fixed number of servers, *M*, as players of the cooperative game, (M, f, ψ), where f is the optimization function and ψ is the Aumann-Shapley mechanism. The target level of f could be state as:

$$f(Z) = \min \sum_{i=1}^{M} \sum_{k=1}^{N} \left(R_{ik} + W_{ik} \right) x_{ik}. \qquad (10.2)$$

The Aumann-Shapley mechanism, ψ, at server i, will be given as:

$$\psi_i = \int_0^1 \frac{\partial f(tZ)}{\partial Z} dt, \qquad (10.3)$$

which may be interpreted as the communication cost imputed to server i. Full distribution of the mechanism requires that:

$$\sum_{i=1}^{M} Z\psi_i = f(Z). \qquad (10.4)$$

Now, each sever would incur a communication cost equal to $Z\psi_i$ due to the accesses made to the data objects hosted by that server. In order to bring a meaning to the Aumann-Shapley mechanism, we need to solve DRP in conjunction to the Aumann-Shapley mechanism. This can be done very efficiently by taking the Lagrangian of RPP.

However, the Lagrangian function on DRP in conjunction with the Aumann-Shapley mechanism using non-linear programming methods would generate multiple solutions, which is not what we desire. To negate the problem of multiple solutions, we take the Lagrangian on the dual of RPP, Z_D, in conjunction with the Aumann-Shapley mechanism, $Z_D(\psi)$, and we get:

$$Z_D(\psi) = \min \sum_{i=1}^{M} \sum_{k=1}^{N} \left(R_{ik} + W_{ik}\right) x_{ik} + \sum_{i=1}^{M} \psi_i \left(\sum_{k=1}^{N} o_k x_{ik} - s_i\right). \quad (10.5)$$

For simplicity, $Z_D(\psi)$ can be written as:

$$Z_D(\psi) = \min \sum_{k=1}^{N} \left(\sum_{i=1}^{M} \left(\left(R_{ik} + W_{ik}\right) + \psi_i o_k\right) x_{ik}\right) - \sum_{i=1}^{M} \psi_i s_i, \quad (10.6)$$

Then the Lagrangian dual problem is as follows:

$$Z_{LD} = \max_{\psi \geq 0} (Z_D(\psi)). \quad (10.7)$$

For a fixed, ψ, (10.5) can be decomposed into sub-problems each of which corresponds to individual server's communication cost as illustrated in (10.4). Each sub-problem is a bounded variable knapsack problem. (This fact is inline with the initial proof of NP-hardness of the data placement problem, where the authors in [77] showed a reduction to the binary knapsack problem.) These sub-problems can easily be solved by a dynamic programming algorithm with a running time $O(M^3N^2)$ [75]. However, the algorithm works only when the lower bound on every variable is 0. This is certainly not the case with DRP and as discussed in earlier the Aumann-Shapley mechanism has continuous first partial derivatives; therefore, the lower bounds on some variables may be positive. For this purpose, we need to devise a technique that can cater for the possible positive lower bounds and which is original to this research.

To find Z_{LD}, we need to find a ψ which gives a maximum of $Z_D(\psi)$ over all $\psi_i \geq 0$. For this purpose we make use of the celebrated sub-gradient method coupled with a branch-and-bound technique to prune and refine the sub-gradient method. Now suppose we are given a current ψ^t at iteration t and an optimal replica placement x_{ik}^t to $Z_D(\psi^t)$, the next step is decided by:

$$\psi^{t+1} = \max\left\{0, \psi^t + \alpha^t\left(\sum_{k=1}^{N} o_k x_{ik} - s_i\right)\right\}, \tag{10.8}$$

$$\text{where } \alpha^t = \left(Z^* - Z_D\left(\psi^t\right)\right) \Big/ \sum_{i=1}^{M}\left(\sum_{k=1}^{N} o_k x_{ik} - s_i\right)^2, \tag{10.9}$$

and Z^* is the objective value of the best known feasible solution to DRP. We set the stopping criteria for the gradient method at iteration t as follows:

After a specified number of iterations, (a)

$Z_D(\psi^t) \geq Z^* - 1$, (b)

$\sum_{k=1}^{N} o_k x_{ik} \leq s_i, \forall i, (1 \leq i \leq M)$, (c)

$x_{p_k k} = 1, \forall P_k, \forall k, (1 \leq k \leq N)$, (d)

Note that we have used criterion (b) instead of $Z_D(\psi^t) \geq Z^*$ since the Lagrangian generated costs are integral. Criteria (c) and (d) represent the optimality conditions. The whole process of the COOP technique is presented in Figure 10.1.

Now suppose that case (b) does not occur within the iteration limit (50 in our implementation). If the current solution satisfies (c) we have found a new feasible

The COOP Technique

```
1   $x^0_{ik} \leftarrow x^*_{ik}$
2   If $\partial = \varnothing$ then **goto** step 5
3   Select $i \in M$
4   $\partial \leftarrow \partial$ - {$i$}
5   $x^0_{ik} \leftarrow x^0_{ik} + INC(i)$
6   Set $w = |\sigma|$; Set $t = 0$
7   Select $i \in M$                                  /* This i is different form the one in Step 3 */
8   $t = t + 1$
9   $x^0_{ik} \leftarrow x^0_{ik} - DEC(i)$
10  If $\sum_{k=1}^{N} o_k x^0_{ik} \le s_i$ , then $\sigma \leftarrow \sigma$ - {$i$}
11  If $\sigma = \varnothing$ then **exit**                        /* Solution is found */
12  If $t < w$  the **goto** step 7
13  If $\partial = \varnothing$ then **exit** else **goto** step 5      /* In case of exit the solution is not found */
```

***INC*(*i*)**

```
1   Set $\beta_i = s_i - \sum_{k=1}^{N} o_k x_{ik}$
2   Compute ($R_{ik} + W_{ik}$) for $O_k$, $\forall k$, ($1 \le k \le N$)
    and store them in set $C$
3   Pick $O_k$ from $C$ as argmax{$C$} and delete $O_k$ from $C$
4   If $\sum_{k=1}^{N} o_k \le \beta_i$ then $w \leftarrow k + 1$ else **goto** step 3
5   Output $x_{ik} = \begin{cases} o_k & k < w \\ \beta_i - \sum_{K<w} o_k & k = w \\ 0 & k > w \end{cases}$
```

***DEC*(*i*)**

```
1   Set $\gamma_i = \sum_{k=1}^{N} o_k x_{ik} - s_i$
2   Compute ($R_{ik} + W_{ik}$) for $O_k$, $\forall k$, ($1 \le k \le N$)
    and store them in set $C$
3   Pick $O_k$ from $C$ as argmin{$C$} and delete $O_k$ from $C$
4   If $\sum_{k=1}^{N} o_k \le \gamma_i$ then $w \leftarrow k + 1$ else **goto** step 3
5   Output $x_{ik} = \begin{cases} o_k & k < w \\ \gamma_i - \sum_{k<w} o_k & k = w \\ 0 & k > w \end{cases}$
```

Figure 10.1: The pseudo-code for the COOP procedure.

solution. Then we update Z^* and continue the sub-gradient iterations. Suppose (d) as well as (c) occur, *i.e.*, the optimality conditions hold within the iteration limit. If the current node is the root node of the branch-and-bound tree, the algorithm ends with the solution x_{ik}, an optimal solution to RPP. Otherwise, Z^* is updated and we continue the sub-gradient iterations. On the other hand, if the optimality conditions do not hold within the iteration limit, we branch at that node and generate two or more child nodes to further improve on the result. We set $\psi^0 = 0$ at the root node and use the Lagrangian at the parent node as the initial value at the child node to avoid unnecessary computations at the child node as suggested in [5]. The selection of the next node to solve is based on the best bound rule.

10.2.4 Feasibility of the COOP Technique

It rarely happens that the solution to the Lagrangian dual problem is feasible to the original problem. However, it can often be transformed to a feasible solution by a minor modification. A solution to $Z_D(\psi)$ satisfies all constraints but may violate the storage constraint. So we modify the solution so that it satisfies the storage constraint. Let us define:

$$\partial = \left\{ \forall i, \left(1 \le i \le M\right) \mid \sum_{k=1}^{N} o_k x_{ik}^{*} - s_i < 0 \right\}, \tag{10.10}$$

$$\sigma = \left\{ \forall i, \left(1 \le i \le M\right) \mid \sum_{k=1}^{N} o_k x_{ik}^{*} - s_i \ge 0 \right\}, \tag{10.11}$$

where x^*_{ik} is the current infeasible replica placement. Note that ∂ and σ are the set of indices of the storage constraint which are violated by the solution. If we decrease $\sum o_k x^*_{ik}$ for $i \in \sigma$, we may be able to make the solution feasible to the problem represented by the current node. Any decrease of the solution values does not affect the validity of the storage constraint, but a careless decrease may cause the solution to violate some of the primary replica constraint and/or the allocation constraint. On the other hand, if we increase $\sum o_k x^*_{ik}$ for $i \in \partial$, before we decrease $\sum o_k x^*_{ik}$ for $i \in \sigma$, then it is more likely that the modified solution becomes feasible. So the increment and decrement of solution should be determined carefully.

The COOP procedure is initiated by selecting an element in ∂ and proceeds iteratively for the rest of the elements in ∂. For each element of ∂, we select elements in σ iteratively and σ is updated if needed.

We select an element $i \in \partial$ and increase values of the variables that appear in constraint i while keeping the feasibility for all constraints. We call this procedure *INC*(i). Then we select an element $i \in \sigma$ and decrease the values of the variables that appear in constraint i while keeping feasibility for other constraints. We call this procedure *DEC*(i). If *INC*(i) succeeds in making the constraint i feasible, then we delete i from ∂. We perform *INC*($\cdot$) for the rest of the elements in ∂. We repeat the process for the rest of the elements i using *DEC*($\cdot$). The order of selecting elements in ∂ and σ is arbitrary. The objective of *INC*(i) is to maximize β_i - $\sum_{\mathrm{k<w}} o_k$. If x_{ik} is obtained after performing *INC*(i), the solution from x^*_{ik} is changed into x_{ik}. On the other hand, the objective of *DEC*(i) is to maximize γ_i - $\sum_{\mathrm{k<w}} o_k$. If x_{ik} is obtained after performing *DEC*(i), the solution from x^*_{ik} is changed into x_{ik}.

10.2.5 Branching Rules of the COOP Technique

We consider three different branching rules when we branch at a node in the branch-and-bound tree. Let x^*_{ik} be the solution obtained at the current node of branch-and-bound tree and let x_{ik} be the selected variable for branching. We denote the lower and upper bounds of the variable x_{ik} by l_{ik} and u_{ik}, respectively.

First, we consider a rule (Rule 1) in which the variable has a fixed value at each generated node. In this rule, $(u_{ik} + 1)$ nodes are generated and the values of the selected variable in the nodes are set to l_{ik} (= 0) through u_{ik}, respectively. There are no special priorities in selecting the variables used to branch.

Another branching rule (Rule 2) is based on a dichotomy branching strategy. In this rule, the variable that has the largest gap between the current upper and lower bounds is selected. This rule generates two nodes, a node with $l_{ik} \leq x_{ik} \leq \lfloor(u_{ik} + l_{ik})/ 2\rfloor$ and the other node with $\lfloor(u_{ik} + l_{ik})/ 2\rfloor + 1 \leq x_{ik} \leq u_{ik}$.

Finally, the third rule (Rule 3) is based on the dichotomy branching strategy considering the current solution. The variable that has the largest gap between the current upper and lower bounds in the most violated constraint is selected for branching. If the selected variable has value x_{ik} , u_{ik}, this rule generates two nodes, a node with $l_{ik} \leq x_{ik} \leq x^*_{ik}$ and the other node with $x^*_{ik} + 1 \leq x_{ik} \leq u_{ik}$. In case that we have a solution with $x^*_{ik} = u_{ik}$, we branch the node with $l_{ik} \leq x_{ik} \leq x^*_{ik} - 1$ and $x_{ik} = x^*_{ik}$.

10.3 Experimental Results

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory. COOP was implemented using Ada and Ada GNAT's distributed systems annex GLADE [97].

To establish diversity in our experimental setups, the network connectively was changed considerably. We used GT-ITM for the network topologies, the procedure for which is as follows: A random graph $G(M,P(\text{edge} = p))$ with $0 \leq p \leq 1$ contains all graphs with nodes (servers) M in which the edges are chosen independently and with a probability p. The pure random topologies were obtained with $p = \{0.4, 0.5, 0.6, 0.7, 0.8\}$. In each of these topologies the distance between two serves was reversed mapped to the communication cost of transmitting a 1kB of data and the latency on a link was assumed to be 2.8×10^{-8} m/s (copper wire).

To evaluate the replica allocation methods under realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 web server. Each experimental setup was evaluated thirteen times, *i.e.*, only the Friday (24 hours) logs from May 1, 1998 to July 24, 1998. (The Friday logs have the heaviest traffic compared to any other day of the week.) To process the logs, we wrote a script that returned: only those objects which were present in all the logs (25,000 in our case), 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 five hundred clients (maximum experimental setup). 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 was in the range of 1-2 million. The primary replicas' original server was mimicked by choosing random locations. The capacities of the servers C% were generated randomly with range from *Total Primary Object Sizes/2* to *1.5×Total Primary Object Sizes*. The variance in the object size collected from the access logs helped to instill enough miscellanies to benchmark object updates. The updates were randomly pushed onto different servers, 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.

Since the access logs are of the year 1998, we first use Inet [17] topology generator to estimate the number of nodes in the network. This number came up to be 3718, *i.e.*, there were 3718 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 at $M = 3718$.

Comparative algorithms: For comparison, we chose three types of replica allocation methods. To provide a fair comparison, the assumptions and system parameters were kept the same in all the methods. For the data replication problem, the techniques proposed in [57], [75], [78] and [100] are the only ones that address the problem domain similar to ours. We select from [100] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [75]); thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [63] (the efficient branch and bound based technique Aε-Star) and [78] (the genetic algorithm based method GRA) are also among the chosen techniques for comparisons. We encourage the readers to obtain an insight on the comparative techniques from the referenced papers.

Performance metric: The solution quality was measured in terms of total communication cost (OTC percentage) that was saved under the replica scheme found by the replica placement methods, compared to the initial one, *i.e.*, when only primary copies exists.

Comparative analysis: We observe the effects of increase in storage 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. This is observable in Figure 10.2, which shows the performance of the algorithms. GRA performed the worst. COOP and Greedy showed an immediate initial increase (the point after which further replicating objects is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most OTC savings (57%) followed by Greedy with 44%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this study due to space restrictions) that the increase in capacity from 10% to 19%, resulted in 4.7 times (on average) more replicas for all the algorithms.

Next, we observe the effects of increase in the read and write frequencies. Since these two parameters are complementary to each other, we describe them together. To observe the system utilization with varying read/write frequencies, we kept the number of servers and objects 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 in the system requires the replicas be placed as close as to the primary server as possible (to reduce the update broadcast).

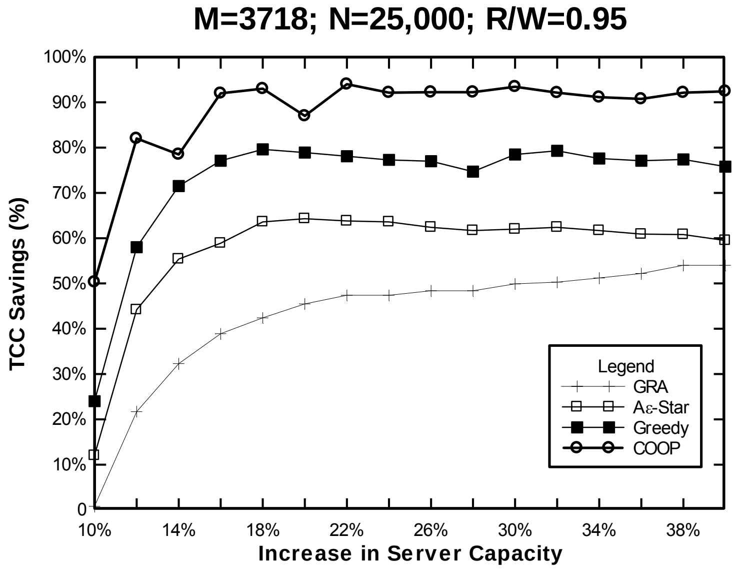

Figure 10.2: OTC savings versus capacity.

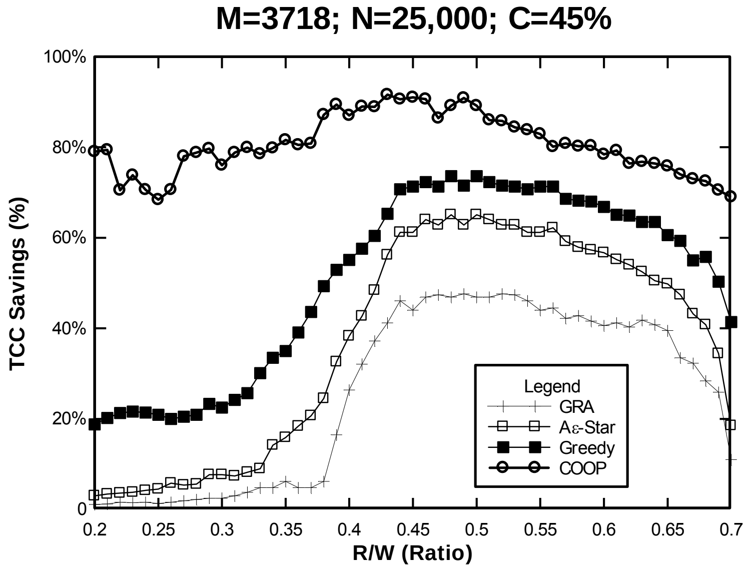

Figure 10.3: OTC savings versus read/write ratio.

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

Table 10.1: Running time of the replica placement methods in seconds for small problem instances [*C*=20%, *R/W*=0.45]

Problem Size	Greedy	GRA	Aε-Star	COOP
M=200, *N*=500	84.13	111.19	116.61	**64.59**
M=200, *N*=1000	91.90	115.68	123.56	**58.30**
M=200, *N*=1500	93.91	121.21	136.62	**60.87**
M=300, *N*=500	114.28	152.30	168.93	118.14
M=300, *N*=1000	131.00	150.04	178.59	134.61
M=300, *N*=1500	162.25	178.30	215.68	196.75
M=400, *N*=500	151.68	184.95	238.52	**149.92**
M=400, *N*=1000	161.58	202.17	284.00	196.81
M=400, *N*=1500	169.29	245.31	324.75	175.65

Table 10.2: Running time of the replica placement methods in seconds for large problem instances [*C*=45%, *R/W*=0.85]

Problem Size	Greedy	GRA	Aε-Star	COOP
M=2500, *N*=15,000	310.14	491.00	399.63	**211.64**
M=2500, *N*=20,000	330.75	563.25	442.66	339.12
M=2500, *N*=25,000	357.74	570.02	465.52	370.38
M=3000, *N*=15,000	452.22	671.68	494.60	556.98
M=3000, *N*=20,000	467.65	726.75	498.66	**341.61**
M=3000, *N*=25,000	469.86	791.26	537.56	549.38
M=3718, *N*=15,000	613.27	883.71	753.87	742.70
M=3718, *N*=20,000	630.39	904.20	774.31	**629.67**
M=3718, *N*=25,000	646.98	932.38	882.43	654.33

between good and marginal algorithms. The plot in Figure 10.3 shows the results of read/write ratio against the OTC savings. A clear classification can be made between the algorithms. COOP and Greedy incorporate the increase in the number of reads by replicating more objects and thus savings increased up to 92%, while GRA gained the least of the OTC savings of up to 42%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial selection of gene population (for details see [78]). Moreover, GRA maintains a localized network perception. 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, COOP, Aε-Star and Greedy never tend to deviate from their global view of the problem.

Lastly, we compare the termination time of the algorithms. Various problem instances were recorded with *C* = 20%, 45% and *R/W* = 0.45, 0.85. The entries in Tables 10.1 and 10.2 made bold represent the fastest time recorded over the problem instance. It is observable that Greedy terminated faster than all the other techniques, followed by COOP, Aε-Star, and GRA.

In summary, based on the solution quality alone, the replica allocation methods can be classified into four categories: 1) High performance: COOP; 2) Medium-High performance: Greedy; 3) Medium performance: Aε-Star; 5) Low performance: GRA. Considering the execution time, Greedy and COOP did extremely well, followed by Aε-Star and GRA.

10.4 Concluding Remarks

This chapter proposed a cooperative game theoretical replica placement technique (COOP) for object based data replication in large distributed computing systems. COOP is a protocol for automatic replication of objects in response to demand changes. It aims to place objects in the proximity of a majority of requests while ensuring that no hosts become overloaded.

The proposed COOP technique improved the performance relative to other conventional methods in four ways. First, the number of replicas in a system was controlled to reflect the ratio of read versus write access. To maintain concurrency control, when an object is updated, all of its replicas need to be updated simultaneously. If the write access rate is high, there should be few replicas to reduce the update overhead. If the read access rate is overwhelming, there should be a high number of replicas to satisfy local accesses. Second, performance was improved by replicating objects to the servers based on locality of reference. This increases the probability that requests can be satisfied either locally or within a desirable amount of time from a neighboring server. Third, replica allocations were made in a fast algorithmic turn-around time.

CHAPTER 11

FUTURE DIRECTIONS, VIEWS, AND VISIONS

Our discussion only encircled the game theoretical auctions that had a central body to collect the information from the players, and based on that conclude a decision. However, there maybe systems, such as, grid computing and P2P system, that explicitly require a fully distributed mechanism. For instance, consider grid computing, which is predominately concerned with coordinated resource sharing in a dynamic, and sometimes in multi-organization structure. Consider also the P2P systems, which are similar to the Grids but characteristically have more users with a wide spectrum of capabilities. Grids and P2P systems have distinct characteristics and stakeholders that require very efficient and effective resource allocation mechanisms, but there is no one central decision making body. Thus, we need to consider applying distributed game theoretical auction mechanism, or the likes of it which can consider to implement a social choice function under the constraint that no central decision making body computes the outcome. This need can be due to:

1. The system has a structure which does not allow a central resource manager.
2. The system requires every entity to be a self sufficient.

A distributed game theoretical auction mechanism would exhibit among others the following advantages over an ordinary game theoretical auction mechanism, and is a strong candidate for resource allocation and management techniques in grid and P2P computing:

1. A distributed game theoretical auction mechanism would transfer the computational workload from a central decision making body in the mechanism to the players.
2. A distributed game theoretical auction mechanism would bring in robustness to the system, since in an ordinary game theoretical auction mechanism the communications between the players and the central decision making body are critical, and their malfunctioning may incapacitate the system. In distributed game theoretical auction mechanism this communication structure simply does not exist, but at a cost – the system may only be able to attain suboptimal results.
3. Since in a distributed game theoretical auction mechanism no single entity would compute the outcome, a higher degree of trust would exist in the system.
4. Due to its distributed nature, the communication would never converge to a single point, thus, there would be no bottlenecks.

Briefly, a distributed game theoretical auction mechanism would distribute the mechanism's rules across the players so that they can perform computations (and eventually reach to an outcome) based on the message sent and received from players in the system. Although, this setting is intriguing, yet it posses several challenges, which we enumerate as follows:

1. The grand challenge here would be to make these players play in a selfless manner, since they now have a firm control over the distributed structure of the underlying auction mechanism.

2. Another grand problem would be to reduce the complexity of messages passing in the communication network.
3. Computationally, we would seek to find social choice functions that can actually converge to solutions in a fully distributed fashion – something on the line of the distributed Vickrey auction implementation, which is a classical example of a canonically distributed convergence.
4. Theoretically, one needs to seek that the strategies applied by the players cater for cartel type behaviors. For instance, imagine a P2P system in which some serves only selectively (on personal preference) allow the sharing or resources. That kind of behavior has to be suppressed at all costs.

APPENDIX A

PUBLICATIONS

PUBLICATIONS

Book Chapters

S. U. Khan and I. Ahmad, "Game Theoretical Solutions for Data Replication in Distributed Computing Systems," in *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Eds., Chapman & Hall/CRC Press, Boca Raton, FL, USA, 2007, ISBN 1-584-88623-4. (In press.)

Journal Papers

S. U. Khan and I. Ahmad, "Replicating Data Objects in Large Distributed Computing Systems: An Axiomatic Game Theoretical Mechanism Design Approach," *IEEE Transactions on Parallel and Distributed Systems*, 2007. (Submitted.)

S. U. Khan and I. Ahmad, "Replicating Data Objects in Large-scale Distributed Computing Systems using Game Theoretical Auctions," *Journal of Parallel and Distributed Computing*, 2007. (Submitted.)

S. U. Khan and I. Ahmad, "A Game Theoretic Perturbation Mechanism for Object Replication in Large Distributed Systems," *Journal of Parallel and Distributed Computing*, 2007. (Submitted .)

S. U. Khan and I. Ahmad, "Comparison and Analysis of Ten Static Heuristics-based

Internet Data Replication Techniques," *Journal of Parallel and Distributed Computing*, 2007. (Accepted subject to minor revision.)

S. U. Khan and I. Ahmad, "Discriminatory Algorithmic Mechanism Design Based WWW Content Replication," *Informatica*, vol. 31, no. 1, pp. 105-119, 2007.

S. U. Khan and I. Ahmad, "Replicating Data Objects in Large-scale Distributed Computing Systems using Extended Vickery Auction," *International Journal of Computational Intelligence*, vol. 3, no. 1, pp. 14-22, 2006.

Conference Papers

S. U. Khan and I. Ahmad, "A Cooperative Game Theoretical Replica Placement Technique," in *13th International Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, December 2007. (Submitted.)

S. U. Khan and I. Ahmad, "A Semi-Distributed Axiomatic Game Theoretical Mechanism for Replicating Data Objects in Large Distributed Computing Systems," in *21th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, CA, USA, March 2007.

S. U. Khan and I. Ahmad, "A Pure Nash Equilibrium Guaranteeing Game Theoretical Replica Allocation Method for Reducing Web Access Time," in *12th International Conference on Parallel and Distributed Systems (ICPADS)*, Minneapolis, MN, USA, July 2006, pp. 169-176.

S. U. Khan and I. Ahmad, "Data Replication in Large Distributed Computing Systems using Supergames," in *The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, USA, June 2006, pp. 38-44.

S. U. Khan and I. Ahmad, "RAMM: A Game Theoretical Replica Allocation and Management Mechanism," in *8th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, Las Vegas, NV, USA, December 2005, pp. 160-165.

S. U. Khan and I. Ahmad, "Data Replication in Large Distributed Computing Systems using Discriminatory Game Theoretic Mechanism Design," in *8th International Conference on Parallel Computing Technologies (PaCT)*, Krasnoyarsk, Russia, September 2005.

S. U. Khan and I. Ahmad, "A Game Theoretical Extended Vickery Auction Mechanism for Replicating Data in Large-scale Distributed Computing Systems," in *The 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, USA, June 2005, pp. 904-910.

S. U. Khan and I. Ahmad, "A Powerful Direct Mechanism for Optimal WWW Content Replication," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, USA, April 2005, p. 86.

S. U. Khan and I. Ahmad, "N+1st Price Auction Based Replica Schemas," in *1st International Conference on Computational Intelligence*, Istanbul, Turkey, December 2004, pp. 256-259.

S. U. Khan and I. Ahmad, "Heuristics-based Replication Schemas for Fast Information Retrieval over the Internet," in *17th International Conference on Parallel and Distributed Computing Systems (PDCS)*, San Francisco, CA, USA, September 2004, pp. 278-283.

REFERENCES

[1] G. Abdulla, *Analysis and Modeling of World Wide Web Traffic*, PhD thesis, Virginia Polytechnic Institute and State University, Virginia, USA, 1998.

[2] T. Abdelzaher and N. Bhatti, “Web Content Adaptation to Improve Sever Workload Behavior,” *Computer Networks*, 21(11), pp. 1536-1577, 1999.

[3] I. Ahmad and A. Ghafoor, “Semi-Distributed Load Balancing for Massively Parallel Multicomputer Systems,” *IEEE Trans. Software Engineering*, vol. 17, no. 10, pp. 987-1004, 1991.

[4] V. Almeida, A. Bestavros, M. Crovella and A. de Oliveria, “Characterizing reference locality in the WWW,” in *Proc. of International Conference on Parallel and Distributed Information Systems*, 1996, pp. 92-103.

[5] P. Apers, “Data Allocation in Distributed Database Systems,” *ACM Transactions on Database Systems*, vol. 13, no. 3, pp. 263-304, 1988.

[6] M. Arlitt and 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.

[7] R. Aumann and M. Maschler, “Game Theoretic analysis of a Bankruptcy Problem from the Talmud,” *Journal of Economic Theory* vol. 36, pp. 195-213, 1985.

[8] B. Awerbuch, Y. Bartal and A. Fiat, “Competitive Distributed File allocation,” in *Proc. 25th ACM STOC*, Victoria, B.C., Canada, 1993, pp. 164-173.

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

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

[11] S. Baker and B. Moon, “Scalable Web Server Design for Distributed Data Management,” in *Proceedings of the 15th International Conference on Data Engineering*, 1999, p. 86.

[12] S. Bradley, A. Hax, and T. Magnanti, *Applied Mathematical Programming*, Addison-Wesley, Reading, Massachusetts, USA, 1977.

[13] L. Breslau, P. Cao, L. Fan, G. Philips and S. Shenker, “Web caching and Zipf-like distributions: Evidence and implications,” in *Proc. of IEEE INFOCOM*, 1999, pp. 126-134.

[14] K. Calvert, M. Doar, E. Zegura, “Modeling Internet Topology,” *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160-163, 1997.

[15] R. Casey, “Allocation of Copies of a File in an Information Network,” *in Proc. Spring Joint Computer Conf.*, IFIPS, 1972, pp. 617-625.

[16] C. Ceri, G. Pelagatti, and G. Martella, “Optimal File Allocation in a Computer Network: A Solution based on Knapsack Problem,” *Computer Networks*, vol. 6, pp. 345-357, 1982.

[17] H. Chang, R. Govindan, S. Jamin and S. Shenker, "Towards Capturing Representative AS-Level Internet Topologies," *Computer Networks Journal*, vol. 44,

no. 6, pp 737-755, 2004.

[18] M. Charikar, S. Guha, E. Tardos and 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.

[19] K. Chandy and J. Hewes, "File Allocation in Distributed Systems," in Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, 1976, pp. 10-13.

[20] L. Chen and 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.

[21] W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Transactions on Computers*, vol. 18, no. 10, pp. 885-889, 1969.

[22] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. Papadimitriou and J. Kubiatowicz, "Selfish Caching in Distributed Systems: A Game-Theoretic Analysis," in *Proc. of 23rd ACM Symposium on Principles of Distributed Computing*, 2004, pp. 21-30.

[23] I. Cidon, S. Kutten, and R. Soffer, "Optimal Allocation of Electronic Content," in *Proc. of IEEE INFOCOM*, April 2001,pp. 1773-1780.

[24] E. Clarke, "Multipart Pricing of Public Goods," *Public Choice*, vol. 11, pp. 17-33, 1971.

[25] V. Conitzer and T. Sandholm, "Complexity of Mechanism Design," in *Proc. of International Conference on Uncertainty in Artificial Intelligence*, 2002, pp. 103-110.

[26] S. Cook, J. Pachl, and I. Pressman, “The Optimal Location of Replicas in a Network using a READ-ONE-WRITE-ALL Policy,” *Distributed Computing*, vol. 15, no. 1, pp. 57-66, 2002.

[27] R. Dash, N. Jennings, and D. Parkers, “Computational Mechanism Design: A Call to Arms,” *IEEE Intelligent Systems*, vol. 18, no. 6, pp. 40-47, 2003.

[28] B. Davison, “A Survey of Proxy Cache Evaluation Techniques,” in *Proceedings of the 4th International Web Caching Workshop*, 1999, pp. 67-77.

[29] G. Demange, D. Gale and M. Sotomayor, “Multi-item auctions,” *Journal of Political Economy* , no. 94, 1986, pp. 836-872.

[30] L. Dowdy and D. Foster, “Comparative Models of the File Allocation Problem,” *ACM Computing Surveys*, vol. 14, no. 2, pp. 287-313, 1982.

[31] K. Eswaran, “Placement of Records in a File and File Allocation in a Computer Network,” in *Proceedings of IFIP Congress*, 1974, pp. 304-307.

[32] A. Fabrikant, C. Papadimitriou and K. Talwar, "The Complexity of Pure Nash Equilibria," in *Proc. of 36th ACM SToC*, 2004, pp. 604-612.

[33] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator and N. Young, “Competitive Paging Algorithms,” *Journal of Algorithms*, vol. 12, no. 4, pp. 685-699, 1991.

[34] S. Floyd and V. Paxson, “Difficulties in Simulating the Internet,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 253-285, 2001.

[35] M. Garey and D. Johnson, *Computers and Intractability*, W.H. Freeman and Co., Murray Hills, New Jersey, USA, 1979.

[36] N. Gautam, “An Integer Programming Formulation of the Web Server Location

Problem," Available at: http://ie.tamu.edu/people/faculty/Gautam/papers/gweb.pdf.

[37] R. Gonen and 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.

[38] J. Green and J. Laffont, "Characterization of Satisfactory Mechanisms for the revelation of Preferences for Public Goods," *Econometrica*, pp. 427-438, 1977.

[39] D. Grosu and A. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems," *IEEE Trans. Systems, Man and Cybernatics B*, 34(1), pp. 77-84, 2004.

[40] T. Groves, "Incentives in Teams," *Econometrica*, vol. 41, pp. 617-631, 1973.

[41] P. Habegger and 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.

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

[43] A. Heddaya and S. Mirdad, "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents," *in Proc. 17th International Conference on Distributed Computing Systems*, Baltimore, Maryland, 1997, pp. 160-168.

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

[45] K. Jain and 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.

[46] S. Jamin, C. Jin, Y. Jin, D. Riaz, Y. Shavitt and L. Zhang, "On the Placement of Internet Instrumentation," in *Proceedings of the IEEE INFOCOM*, 2000, pp. 295-304.

[47] S. Jamin, C. Jin, T. Kurc, D. Raz and Y. Shavitt, "Constrained Mirror Placement on the Internet," in *Proceedings of the IEEE INFOCOM*, 2001, pp. 31-40.

[48] N. Jennings and S. Bussmann, "Agent-Based Control Systems," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 61-74, 2003.

[49] X. Jia, D. Li, X. Hu, and D. Du, "Optimal Placement of Web Proxies for Replicated Web Servers in the Internet," *The Computer Journal*, vol. 44, no. 5, pp. 329-339, 2001.

[50] S. Jin and A. Bestavros, "Greedydual* Web Caching Algorithm: Exploiting the two sources of Temporal Locality in Web Request Streams," *Computer Communiations*, vol. 24, no. 2, pp. 174-183, 2001.

[51] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, 1998.

[52] J. Kangasharju, J. Roberts and 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.

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

[54] M. Karlsson and 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.

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

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

[57] S. Khan and 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.

[58] S. Khan and 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.

[59] S. Khan and I. Ahmad, "A Game Theoretical Extended Vickery Auction Mechanism for Replicating Data in Large-scale Distributed Computing Systems," in *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications*, 2005, pp. 904-910.

[60] S. Khan and I. Ahmad, "Data Replication in Large Distributed Computing Systems using Discriminatory Game Theoretic Mechanism Design," in *Proc of 8th International Conference on Parallel Computing Technologies*, 2005.

[61] S. Khan and I. Ahmad, "RAMM: A Game Theoretical Replica Allocation and Management Mechanism," in *Proc. 8th International Symposium on Parallel Architectures, Algorithms, and Networks*, 2005, pp. 160-165.

[62] S. Khan and I. Ahmad, "A Pure Nash Equilibrium Guaranteeing Game Theoretical Replica Allocation Method for Reducing Web Access Time," in *Proc. of 12th International Conference on Parallel and Distributed Systems*, 2006.

[63] S. Khan and I. Ahmad, "Replicating Data Objects in Large-scale Distributed Computing Systems using Extended Vickrey Auction," *International Journal of Computational Intelligence*, vol. 3, no. 1, pp. 14-22, 2006.

[64] M. Korupolu, C. Plaxton and L. Rajaraman, "Analysis of a Local Search Heuristic for Facility Location Problems, in *Proc. of the 9th Annual ACM/SIAM Symposium on Discrete Algorithms*, 1998, pp. 1-10.

[65] C. Krick, H. Racke, and M. Westermann, "Approximation Algorithms for Data Management in Networks," in *Proc. of the Symposium on Parallel Algorithms and Architecture*, 2001, pp. 237–246.

[66] V. Krishna. *Auction Theory*, Academic Press, San Diego, U.S.A., 2002.

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

[68] B. Krishnamurthy and J. Wang, "On Network-aware Clustering of Web Clients," in *Proceedings of the ACM SIGCOMM*, 2000, pp. 97-110.

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

[70] Y.-K. Kwok, K. Karlapalem, I. Ahmad and N. Pun, "Design and Evaluation of Data Allocation Algorithms for Distributed Database Systems," *IEEE Journal on*

Selected areas in Communication, vol. 14, no. 7, pp. 1332-1348, 1996.

[71] H. Kuhn, “Excerpt from Montmort's Letter to Nicholas Bernoulli,” in *Precursors in Mathematical Economics: An Anthology*, ser. Reprints of Scarce Works on Political Economy, W. Baumol and S. Goldfeld, eds., vol. 19, pp. 3-6, 1968.

[72] N. Laoutaris, O. Telelis, V. Zissimopoulos and I. Stavrakakis, “Local Utility Aware Content Replication,” to appear in *IFIP Networking Conference*, 2005.

[73] B. Lee and 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.

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

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

[76] Looking Glass Sites Project, Available at: http://www.nanog.org/lookingglass.html.

[77] T. Loukopoulos, *Caching and Replication Schemes on the Internet*, PhD thesis, Hong Kong University of Science and Technology, Hong Kong, China, 2002.

[78] T. Loukopoulos, and I. Ahmad, “Static and Adaptive Distributed Data Replication using Genetic Algorithms,” *Journal of Parallel and Distributed Computing*, vol. 64, no. 11, pp. 1270-1285, 2004.

[79] T. Loukopoulos, I. Ahmad, and D. Papadias, “An Overview of Data Replication

on the Internet," in *Proceedings of the 6th International Symposium on Parallel Architectures Algorithms, and Networks*, pp. 31-36, 2002.

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

[81] B. Maggs, F. Meyer auf der Heide, B. Vocking, and M. Westermann, "Exploiting Locality for Data Management in Systemsof Limited Bandwidth," in *Proc. of the Symposium on Foundations of Computer Science*, 1997, pp. 284-293.

[82] S. Mahmoud and J. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 66-78, 1976.

[83] S. March and S. Rho, "Allocating Data and Operations to Nodes in Distributed Database Design," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, pp.305-317, 1995.

[84] A. Mascolell, M. Whinston, and J. Green, *Microeconomic Theory*, Oxford University Press, 1995.

[85] R. McAfee and J. McMillan, "Actions and Bidding," *Journal of Economics Literature*, vol. 25, pp. 699-738, 1987.

[86] A. Medina, I. Matta and J. Byers, "On the Origin of Power Laws in Internet Topologies," *ACM Computer Communication Review*, vol. 30, no. 2, pp. 18-28, 2000.

[87] F. Meyer auf der Heide, B. Vocking, and M. Westermann,"Caching in Networks," in *Proc. of the 11th ACM-SIAM Symposium On Discrete Algorithms*, 2000, pp. 430–

439.

[88] R. Myerson, *Game Theory: Analysis of Conflict*, Harvard University Press, 1997.

[89] B. Narebdran, S. Rangarajan and 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.

[90] NASA Keneddy Space Center access log, Available at: http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[91] J. Nash, "The Bargaining Problem," *Econometrica*, vol. 18, pp. 155–162, 1950.

[92] J. Nash, “Non-Cooperative Games,” *Annals of Mathematics*, vol. 54, pp. 286-295, 1951.

[93] J. Nash, "Equilibrium Points in N-person Games," in *Proc. of the National Academy of Sciences*, vol. 36, pp. 48-49, 1950.

[94] J. Nash and L. Shapley, "A Simple Three-person Poker Game," *Annals of Mathematical Studies*, vol. 24, pp. 105-106, 1950.

[95] N. Nisan and A. Ronen, “Algorithmic Mechanism Design,” in *Proc. of 31st ACM STOC*, 1999, pp. 129-140.

[96] M. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, 2002.

[97] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and 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.

[98] L. Pautet and S. Tardieu, “GLADE: A Framework for Building Large Object-

Oriented Real-Time Distributed Systems," in *3rd International Symposium on Object-Oriented Real-Time Distributed Systems*, 2000, pp. 244-251.

[99] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, Massachusetts, USA, 1984.

[100] L. Qiu, V. Padmanabhan and G. Voelker, "On the Placement of Web Server Replicas," in *Proc. of the IEEE INFOCOM*, 2001, pp. 1587-1596.

[101] M. Rabinovich, "Issues in Web Content Replication," *Data Engineering Bulletin,* vol. 21, no. 4, pp. 21-29, 1998.

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

[103] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," Internet Engineering Task Force, RFC 1771, 1995.

[104] J. Rosenchein and G. Zolotkin, *Rules of Encounter*, MIT Press, 1994.

[105] T. Sandholm, "Distributed Rational Decision Making," *Multiagent Systems: A modern Approach to Distributed Artificial Intelligence*, G. Weiss, ed., MIT Press, p. 201, 1999.

[106] S. Saurabh and D. Parkes, "Hard-to-Manupilate VCG-Based Auctions," Avaialable at: http://www .eecs. harvard.edu/econcs/pubs/hard_to_manipulate.pdf

[107] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, J. Kubiatowicz, "Maintenance-free Global Storage," *IEEE Internet Computing*, vol. 5, no. 5, pp. 40-49, 2001.

[108] M. Sayal, Y. Breitbart, P. Scheuermann and R. Vingralek, "Selection Algorithms

for Replicated Web Servers," *ACM Performance Evaluation Review*, vol. 26, no. 3, pp. 44-50, 1998.

[109] S. So, I. Ahmad and K. Karlapalem, "Response Time Driven Multimedia Data Objects Allocation for Browsing Documents in Distributed Environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 3, pp. 386-405, 1999.

[110] G. Srinivasan and N. Gautam, "Optimal Location of Web Servers, in *Proc. of the Industrial Engineering Research Conference*, 2002.

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

[112] E. van Damme, Stability and Perfection of Nash Equilibia, Springer-Verlag, 1996.

[113] A. Venkataramanj, P. Weidmann, and M. Dahlin, "Bandwidth Constrained Placement in a WAN," in *Proc. ACM Symposium on Principles of Distributed Computing*, August 2001.

[114] W. Vickrey, "Counterspeculation, Auctions and Competitive Sealed Tenders," *Journal of Finance*, pp. 8-37, 1961.

[115] A. Vigneron, L. Gao, M. Golin, G. Italiano and B. Li, "An Algorithm for Finding a k-median in a Directed Tree," *Information Processing Letters*, vol. 74 , pp. 81-88, 2000.

[116] J. von Neumann, "Zur Theorie der Gesellschaftsspiele," *Mathematische Annalen*, vol. 100, pp. 295-320, 1928.

[117] B. Waxman, "Routing of Multipoint Connections," *IEEE Journal of Selected*

Areas of Communications, vol. 6, no. 9, pp. 1617-1622, 1988.

[118] M. Wellman, "A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problem," *Journal of Artificial Intelligence Research*, vol. 1, pp. 1-23, 1993.

[119] D. White, "An Extension of a Greedy Heuristic for the Knapsack Problem," *European Journal of Operational Research*, vol. 51, pp. 387-399, 1991.

[120] O. Wolfson and S. Jajodia, "Distributed algorithms for dynamic replication of data," in *Proc. ACM Symposium on Principles of Database Systems*, June 1992, pp. 149-163.

[121] O. Wolfson, S. Jajodia and Y. Hang, "An Adaptive Data Replication Algorithm," *ACM Trans. on Database Systems*, 22(4), pp. 255-314, 1997.

[122] E. Zegura, K. Calvert and M. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 770-783, 1997.

[123] G. Zipf, *Human Behavior and the Principle of Least-Effort*, Addison-Wesley, Reading, Massachusetts, USA, 1949.

[124] L. Zhuo, C. Wang and 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.

[125] A. Zoltners, "A Direct Descent Binary Knapsack Algorithm," *Journal of the ACM*, vol. 25, no. 2, pp. 304-311, 1978.

BIOGRAPHICAL INFORMATION

Samee U. Khan received his BS in Computer Systems Engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, in May 1999. He is currently a doctoral candidate in the Computer Science and Engineering Department of the University of Texas, Arlington, TX, USA. 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. Mr. 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.