Discriminatory Algorithmic Mechanism Design Based WWW Content Replication

Samee Ullah Khan and Ishfaq Ahmad
Department of Computer Science and Engineering
University of Texas
Arlington, TX-76019, USA
E-mail: {sakhan, iahmad}@cse.uta.edu

Keywords: data replication, resource allocation, game theory, algorithmic mechanism design, static allocation

Received: November 8, 2005

Replicating data over geographically dispersed web servers reduces network traffic, server load, and more importantly the user-perceived access delays. This paper proposes a unique 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. The mechanism is extensively evaluated against some well-known algorithms, such as: greedy, branch and bound, game theoretical auctions and genetic algorithms. The experimental results reveal that the mechanism provides excellent solution quality, while maintaining fast execution time.

Povzetek: Opisana je metoda za multipliciranje internetnih strani.

1 Introduction

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], [8], 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 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 additional copies. Discussions in [20], [22], [25], [26], [30], etc. 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.

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.

This paper uses 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 this paper, we will apply the derived mechanism to the fine grained data replication problem (DRP) over the Internet. In essence we sculpt the DRP 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.

The major results of this paper are as follows:

1. We derive a general-purpose distributed mechanism that allows selfish agents to compete at each site in the distributed computing system for the rights to replicate objects in a non-cooperative environment.
2. We show that the concurrently played subgames exhibit Nash equilibrium which in turn guarantees Nash equilibrium for the games and the supergame.
3. The mechanism is compared against some well-known techniques, such as: greedy, branch and bound, genetic and game theoretical auctions, employing various internet topology generators and real user access data. The experimental results reveal that the mechanism provides excellent solution quality, while maintaining fast execution time.

This paper is organized as follows. Section 2 formulates the DRP. Section 3 describes the mechanism. The experimental results, related work and concluding remarks are provided in Sections 4, 5 and 6, respectively.

2 Formal Description of the Data Replication Problem

The most frequently used acronyms are recorded in Table 1.

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 [20], [22], [26], [30], etc.). 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 initiate a replica placement algorithm (in our case the mechanism). Note that this time period t is the only parameter that requires human intervention. However, in this paper we use analytical data that enables us to effectively predict the time interval t (see Section 3.4. for details).

Symbols	Meaning
M	Total number of sites in the network.
N	Total number of objects to be replicated.
O_k	k-th object.
o_k	Size of object k.
S^i	i-th site.
s^i	Size of site i.
r_k^i	Number of reads for object k from site i.
R_k^i	Aggregate read cost of r_k^i.
w_k^i	Number of writes for object k from site i.
W_k^i	Aggregate write cost of w_k^i.
NN_k^i	Nearest neighbor of site i holding object k.
$c(i,j)$	Communication cost between sites i and j.
P_k	Primary site of the k-th object.
R_k	Replication schema of object k.
$C_{overall}$	Total overall data transfer cost.
LS	A list of sites that can replicate an object.
L^i	A list of objects that can be replicated onto site S^i.
B_k^i	Benefit of replicating object k onto site S^i.
B_k	Temporary variable to store object valuations.
b^i	Available space at site S^i.
v	Valuation of an agent for an object.
SGRG	Self Generate Random Graphs.
GT-ITM	Georgia Tech Internetwork Topology Models.
GT-ITM PR	GT-ITM Pure Random.
GT-ITM W	GT-ITM Waxman.
SGFCG	Self Generated Fully Connected Graphs.
SGFCGUD	SGFCG Uniform Distribution.
SGFCGRD	SGFCG Random Distribution.
SGRG	Self Generated Random Graphs.
SGRGLND	SGRG Lognormal Distribution.
DRP	Data replication problem.
OTC	Object transfer cost (network communication cost).

Table 1: Notations and their meanings.

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, *i.e.*, $NN_k^i = \{Site\ j \mid j \in R_k \wedge \min c(i,j)\}$. 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 network transfer cost due to object movement, *i.e.* the Object Transfer Cost (OTC). The communication cost of the control messages has minor impact to the overall performance of the system, 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 OTC. The first component of OTC is due to the read requests. Let R_k^i denote the total OTC, 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:

$$R_k^i = r_k^i o_k c\left(i, NN_k^i\right). \quad (1)$$

The second component of OTC is the cost arising due to the writes. Let W_k^i be the total OTC, due to S^is' writing requests for object O_k, addressed to the primary site P^k. This cost is given by the following equation:

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

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 OTC, denoted as $C_{overall}$, due to reads and writes is given by:

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

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:

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

Sites which are not the *replicators* of object O_k create OTC 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:

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

and subject to the primary copies policy:

$$X_{p_k k} = 1 \ \forall \left(1 \leq k \leq N\right).$$

The minimization of $C_{overall}$ will have two impacts on the distributed 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 OTC of the system. In the generalized case, the DRP has been proven to be NP-complete [26].

3 The 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 [18], [29], 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.

3.1 Discriminatory Mechanism

In this paper we limit our analysis to one-shot (single round) mechanisms in which every agent demands a specific entity. Under our DRP 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 paper 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 [16]. 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 [27].

In a discriminatory mechanism, there is no sequential interaction among agents [27]. 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 [12] 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 [17]. Therefore, we are confined to a probabilistic analysis of the discriminatory mechanism.

3.2 Preliminaries

Definition 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 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 [27]. 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 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 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 1 ([17])**:** *Nash equilibrium only depends upon subgame strategy profiles played along the equilibrium path.* ■

Theorem 1 ([16]): *In Nash equilibrium each player's repeated game (supergame) strategy need only be optimal along the equilibrium path.* ■

Remarks – In essence Definitions 3 and 4 and Lemma 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 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 number of games played over time.

3.3 Mechanism Applied to the DRP

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

[1] Define the DRP 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.
2. Establish the fact that subgames conform to Nash equilibrium provided agents play optimally.
3. Use Lemma 1 to establish that the entire game at instance t is in Nash equilibrium.
4. Use Theorem 1 to establish that the entire supergame is in Nash equilibrium.

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.

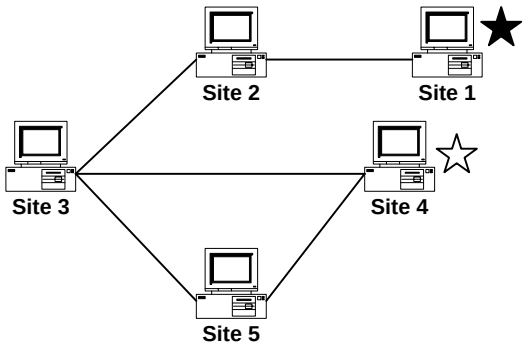

Figure 1(a): The network architecture.

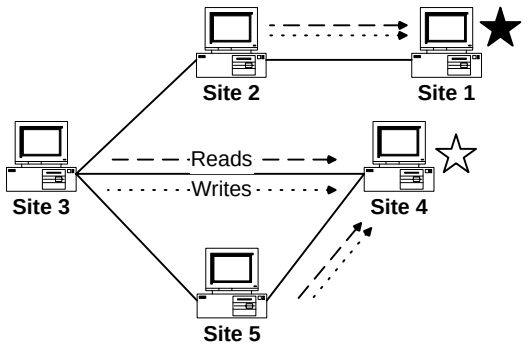

Figure 1(b): Read and write patterns.

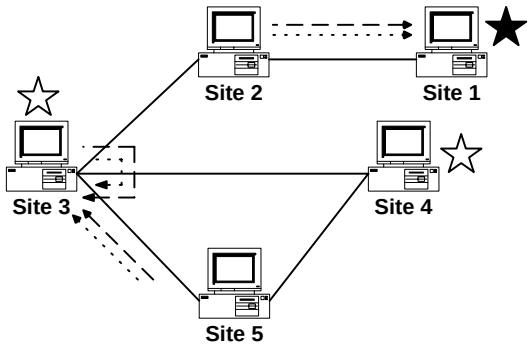

Figure 1(c): Benefits of replication (reads).

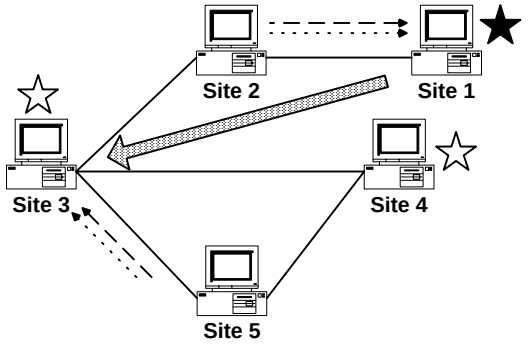

Figure 1(d): Benefits of replication (writes).

2. Game: At any given instance t (t is the instance when a game is invoked, in Section 3.4. we explain what t really means), 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 [26].

3. Subgames: 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.

3.a. 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 $\sum_{k=1}^{n} o_k \leq b^i, n \leq N$. 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. We describe this valuation below.

3.b. 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:

$$B_k^i = R_k^i - \left(\sum_{x=1}^{M} w_k^x o_k c\left(i, P_k\right) - W_k^i \right). \qquad (5)$$

We illustrate the notion of benefit associated with an object k if it is replicated at site i. Figure 1(a) 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 1(b)). Now what would be the benefit of replicating object k at Site 3? In Figure 1(c), 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 1(d)), 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.

3.c. 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) [32], 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.

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 $(F(r))^{N-1}$ which we will denote $F^{N-1}(r)$. 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:

$$u(r,v) = F^{N-1}(r)\left(v - \hat{b}(r)\right). \tag{6}$$

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 6 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:

$$\begin{aligned} &d/dr\left(F^{N-1}(r)\left(v-\hat{b}(r)\right)\right) = \\ &\left(N-1\right)F^{N-2}(r)f(r)\left(v-\hat{b}(r)\right) - F^{N-1}(r)\hat{b}'(r) \end{aligned} \tag{7}$$

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

$$\begin{aligned} &\left(N-1\right)F^{N-2}(v)f(v)\hat{b}(v) + F^{N-1}(v)\hat{b}'(v) = \\ &\left(N-1\right)vf(v)F^{N-2}(v) \end{aligned} \tag{8}$$

Looking closely at the left-hand side of Equation 8, 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 as:

$$d/dv\left(F^{N-1}(v)\hat{b}(v)\right)=(N-1)vf(v)F^{N-2}(v). \quad (9)$$

Now, because Equation 9 must hold for every v, it must be the case that:

$$F^{N-1}(v)b(v)= (N-1)\int_0^v xf(x)F^{N-2}(x)dx+constant. \quad (10)$$

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:

$$\hat{b}(v)=\frac{N-1}{F^{N-1}(v)}\int_0^v xf(x)F^{N-2}(x)dx, \quad (11)$$

which can be written as:

$$\hat{b}(v)=\frac{1}{F^{N-1}(v)}\int_0^v xf(x)F^{N-2}(x)dx. \quad (12)$$

There are two things to notice about the bidding function in Equation 12. 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:

$$\hat{b}(v) = \frac{1}{v^{N-1}}\int_0^v xdx^{N-1} \quad (13)$$

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

$$=\frac{N-1}{v^{N-1}}\int_0^v x^{N-1}dx$$

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

$$=\left(\frac{N-1}{N}\right)v \quad (14)$$

Hence, in conclusion, we have proven the following:

Theorem 2: *If N agents have independent private values drawn from the 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 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 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 1 and Theorem 1. ■

Discriminatory Mechanism

```
Initialize:
01 LS, L^i.
02 WHILE LS ≠ NULL DO
03   PARFOR each S^i ∈ LS DO                    /*M subgames*/
04        FOR each k ∈ O DO
05             B_k = compute (B_k^i × (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 ≥ 0
10    B_k = 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 2: Mechanism game at instance *t*.

We are now ready to present the pseudo-code (Figure 2) 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 4: *In the worst case the mechanism takes* $O(N^2 logN)$ *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 $O(N^2\log N)$. ■

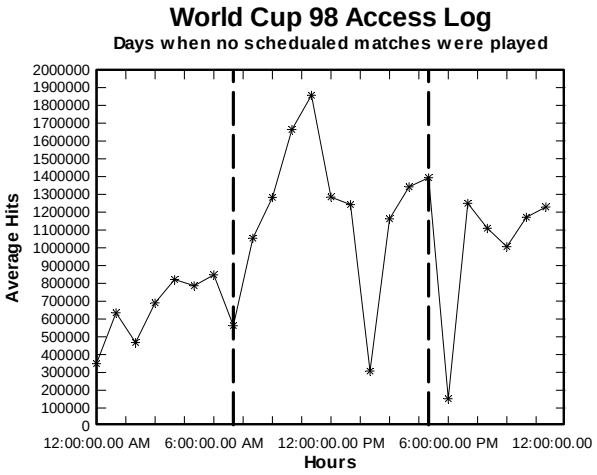

Figure 3(a): Access on days with no matches.

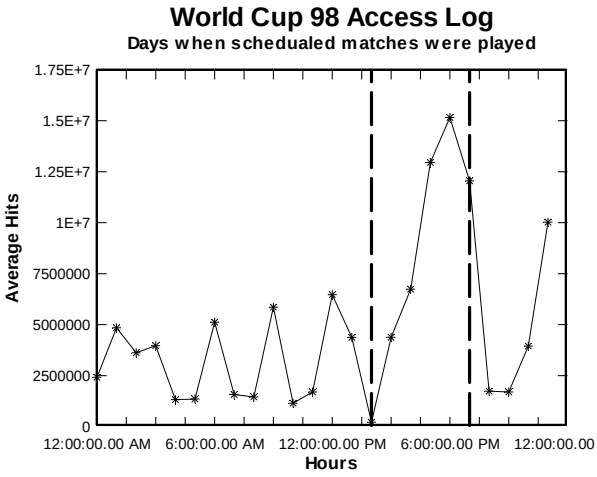

Figure 3(b): Access on days with matches.

3.4 When to invoke the game?

As noted previously (in Sections 2 and 3.3.), the time (interval t) when to initiate the mechanism, *i.e.*, when to play a game at instance t requires high-level human intervention. Here, we will show that this parameter if not totally can at least partially be automated. The decision when to initiate the mechanism depends on the past trends of the user access patterns. The experiments performed to test the mechanism used real user access patterns collected at the 1998 Soccer World Cup website [6]. This access log file has become a default standard over the number of years to benchmark various replica placement techniques. Works reported in [20], [21], [22], [23], [25], and [30] all have used this access log for analysis.

Figures 3(a) and 3(b) show the user access patterns. The two figures represent different traffic patterns, *i.e.*, Figure 3(a) shows the traffic recorded on the days when there was no scheduled match, while Figure 3(b) shows the traffic on the days when there were scheduled matches. We can clearly see that the website incurred soaring and stumpy traffic at various intervals during a 24-hour time period (it is to be noted that the access logs have a time stamp of GMT+1). For example, on days when there was no scheduled match, the traffic was mediocre before 0900 hrs. The traffic increased after 0900 hrs till 2200 hrs. The two vertical dashed lines indicate this phenomenon. These traffic patterns were recorded over a period of 86 days (April 30th 1998 to July 26th 1998). Therefore, on the days when there was no scheduled match, a replica placement algorithm (in our case the mechanism) could be initiated twice daily: 1) at 0900 hrs and 2) at 2200 hrs. The time interval t for 0900 hrs would be t = (2200-0900) = 11 hours and for 2200 hrs would be t = (0900-2200) = 13 hours. On the other hand the days when there were scheduled matches, the mechanism could be initiated at 1900 hrs and 0100 hrs. It is to be noted that the autonomous agents can easily obtain all the other required parameters (for the DRP) via the user access logs and the underlying network architecture.

4 Experimental Setup and the Discussion of Results

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

To establish diversity in our experimental setups, the network connectively was changed considerably. In this paper, we only present the results that were obtained using a maximum of 500 sites (nodes). We used existing topology generator toolkits and also self generated networks. In all the topologies the distance of the link between nodes was equivalent to the communication cost. Table 2 summarizes the various techniques used to gather forty-five various topologies for networks with 100 nodes. It is to be noted that the parameters vary for networks with lesser/larger number of nodes.

To evaluate the chosen replication placement techniques on realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 website [6]. Each experimental setup was evaluated thirteen times, i.e., the Friday (24 hours) logs from May 1, 1998 to July 24, 1998. Thus, each experimental setup in fact represents an average of the 585 (13×45) data set points. To process the logs, we wrote a script that returned: only those objects which were present in all the logs (2000 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 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*

Topology	Mathematical Representation	Parameter Interval Variance
SGRG (12 topologies)	Randomized layout with node degree (d*) and Euclidian distance (d) between nodes as parameters.	d={5,10,15,20}, d*={10,15,20}.
GT-ITM PR [9] (5 topologies)	Randomized layout with edges added between the randomly located vertices with a probability (p).	p={0.4,0.5,0.6,0.7,0.8}.
GT-ITM W [9] (9 topologies)	$P(u,v)=\alpha e^{-d/(\beta L)}$	α={0.1,0.15,0.2,0.25}, β={0.2,0.3,0.4}.
SGFCGUD (5 topologies)	Fully connected graph with uniform link distances (d).	d_1=[1,10],d_2=[1,20],d_3=[1,50], d_4=[10,20], d_5=[20,50].
SGFCGRD (5 topologies)	Fully connected graph with random link distances (d).	d_1=[1,10],d_2=[1,20],d_3=[1,50], d_4=[10,20], d_5=[20,50].
SGRGLND (9 topologies)	Random layout with link distance having a lognormal distribution [15].	μ={8.455,9.345,9.564}, σ={1.278,1.305,1.378}.

Table 2: Parameter interval variance characterization for topologies with 100 nodes.

Sizes. The variance in the object size collected from the access logs helped to install enough diversity to benchmark 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.

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 [22]). For fine-grained replication, the algorithms proposed in [23], [25], [26], and [30] are the only ones that address the problem domain similar to ours. We select from [30] the greedy approach (Greedy) for comparison because it is shown to be the best compared with four other approaches (including the proposed technique in [25]); thus, we indirectly compare with four additional approaches as well. Algorithms reported in [23] (Dutch (DA) and English auctions (EA)) and [26] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons. Due to space limitations we will only give a brief overview of the comparative techniques. Details for a specific technique can be obtained from the referenced papers.

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.

Aε-Star: In [22] 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.

Greedy based technique: We modify the greedy approach reported in [30], to fit our problem formulation. The greedy algorithm 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. The readers will immediately realize that the bidding mechanism reported in this paper works similar 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).

Dutch auction: The auctioneer begins with a high asking price which is lowered until some agent is willing to accept the auctioneer's price. That agent pays the last announced price. This type of auction is convenient when it is important to auction objects quickly, since a sale never requires more than one bid. In no case does the auctioneer reveal any of the bids submitted to him, and no information is shared between the agents. It is shown that for an agent to have a probabilistically superior bid than n-1 other bids; agent should have the valuation divided by n [23].

English auction: In this type of auction, the agents bid openly against one another, with each bid being higher than the previous bid. The auction ends when no agent is willing to bid further. During the auction when an auctioneer receives a bid higher than the currently submitted bids, he announces the bid value so that other agents (if needed) can revise their currently submitted bids. In [23], the discussion on EA reveals that the optimal strategy for a bidder i is to bid a value which is directly derived from his valuation.

GRA: In [26], 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.

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 (Section 4.3.) 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 4), by setting the number of objects to 2000, while in Figure 5, 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 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 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 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 [22] and [26].

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 [26] and [30]. But there the authors had used a uniformly distributed link cost topology, and their traffic was based on the Zipf distribution [33]. 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 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 6, 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 paper 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.

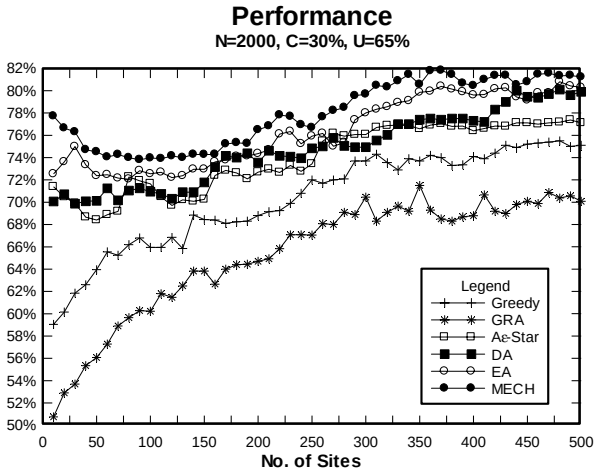

Figure 4: OTC savings versus number of sites.

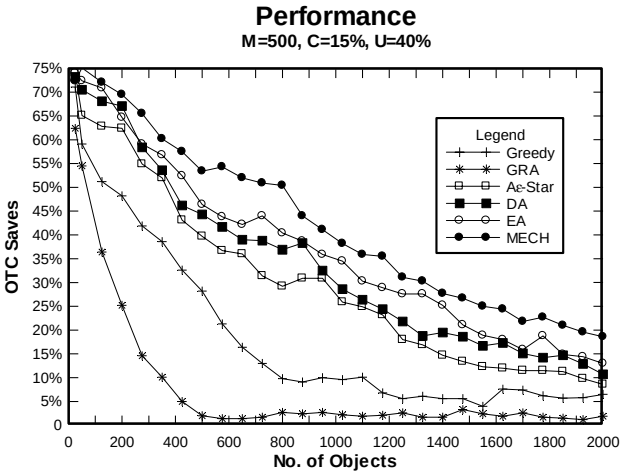

Figure 5: OTC savings versus number of objects.

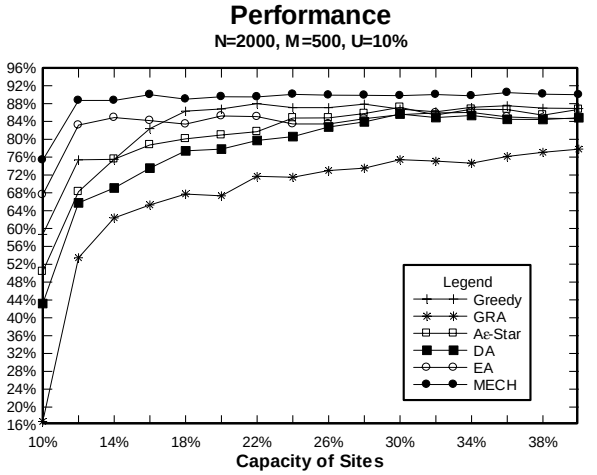

Figure 6: OTC savings versus capacity.

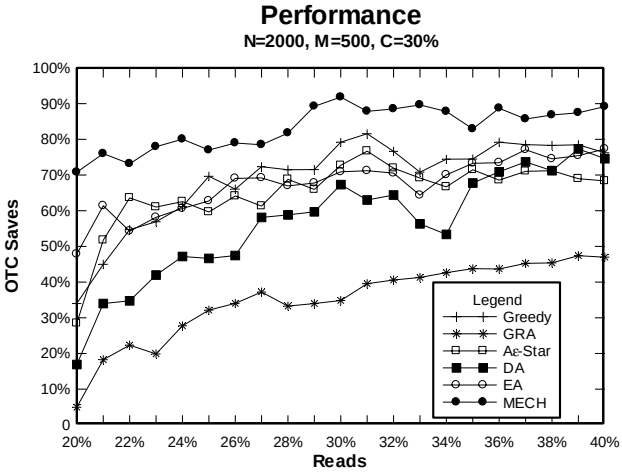

Figure 7: OTC savings versus reads.

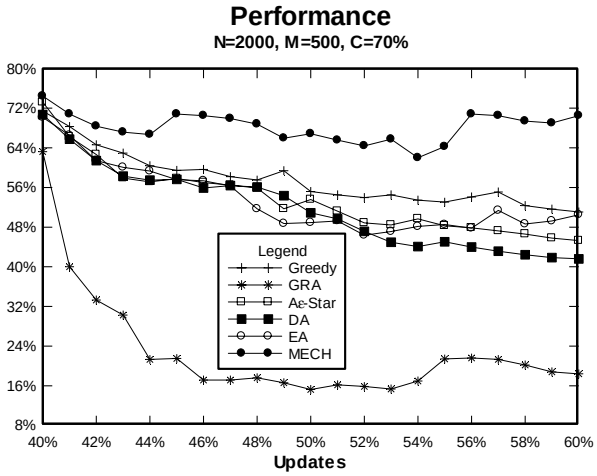

Figure 8: OTC savings versus updates.

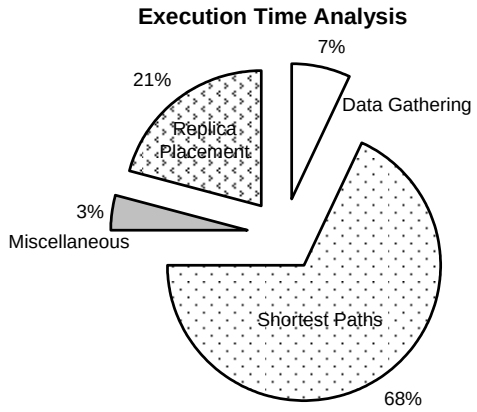

Figure 9: Execution time components.

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 7 and 8 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 [26]). 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 9. 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 3 and 4.

Various problem instances were recorded with C =

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 3: Running time in seconds [C=20%, U=25%].

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 4: Running time in seconds [C=35%, U=35%].

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**

Table 5: Average OTC (%) savings under some problem instances.

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 5 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.

4.3 Comparative Supergame Analysis

Here, we present some supplementary results regarding the supergame that strengthen our comparative analysis claims provided in Section 4.2. We show the relative performance of the techniques with load and storage capacity variance. The plots in Figures 10-13 show the recorded performances. All the plots summarize the measured performance with varying parameters observed over a time period of 86 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

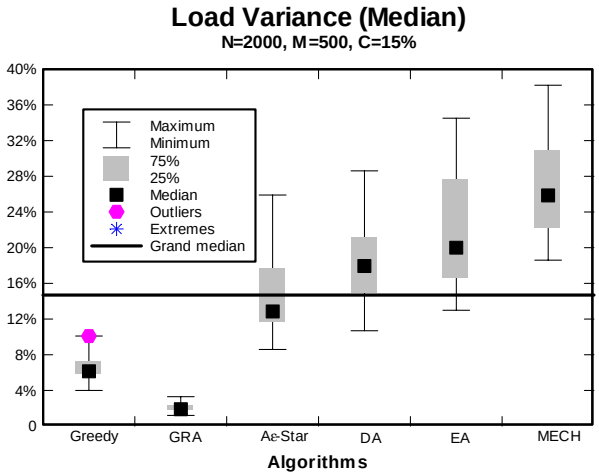

Figure 10: Median load variance.

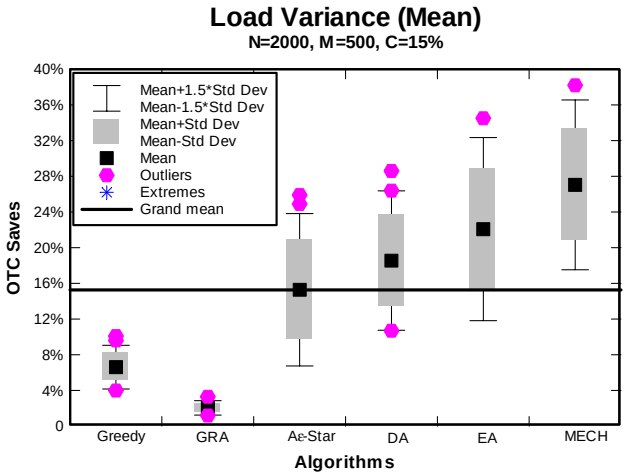

Figure 11: Mean load variance.

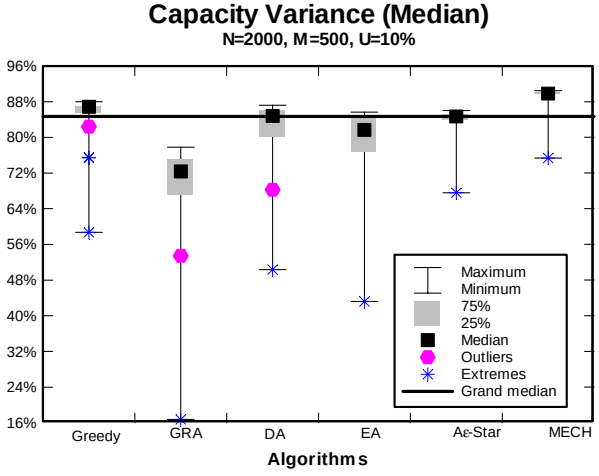

Figure 12: Median capacity variance.

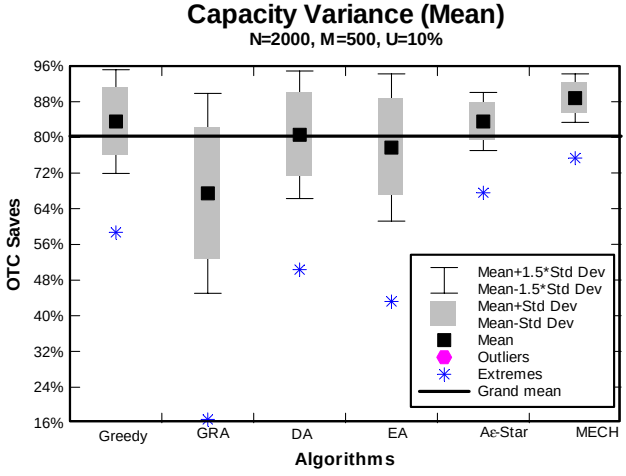

Figure 13: Mean capacity variance.

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 10 and 11 show mediocre OTC savings, simply because they encapsulated the performance of the networks where update ratio was extremely high.

5 Related Work

The data replication problem as presented in Section 2 is an extension of the classical file allocation problem (FAP). Chu [11] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [10] extended this work by distinguishing between updates and read file requests. Eswaran [14] proved that Casey's formulation was NP complete. In [28] 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 [13]. Apers in [4] considered the data allocation problem (DAP) in distributed databases where the query execution strategy influences allocation decisions. In [24] the authors proposed several algorithms to solve the data allocation problem in distributed multimedia databases (without replication), also called as video allocation problem (VAP).

Most of the research papers outlined in [13] aim at formalizing the problem as an optimization one, sometimes using multiple objective functions. Network traffic, server throughput and response time exhibited by users are considered for optimization. Although a lot of effort was devoted in providing comprehensive models, little attention has been paid to good heuristics for solving this complex problem. Furthermore access patterns are assumed to remain static and solutions in the dynamic case are obtained by re-executing a time consuming mathematical programming technique.

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 [33] 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. [19] proposed protocols that load balance the workload among replicas. In [31], Rabinovich et al. proposed a protocol for dynamically replicating the contents of an ISP (Internet Service Provider) in order to improve client-server proximity without overloading any of the servers. However updates were not considered.

Our work differs from all the above in: 1) Taking into account the more pragmatic scenario in today's distributed information environments, we tackle the case of allocating replicas so as to minimize the network traffic under storage constraints with "read from the nearest" and "update through the primary server" policies, and 2) in using game theoretical techniques.

6 Concluding Remarks

This paper proposed a game theoretical discriminatory mechanism (MECH) for fine-grained data replication in large-scale distributed computing systems (e.g. the Internet). 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.

References

[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 Trans. Database Systems*, 13(3), pp. 263-304, 1988.

[6] M. Arlitt and T. Jin, "Workload characterization of the 1998 World Cup Web Site," Tech. report, Hewlett Packard Lab, Palo Alto, HPL-1999-35(R.1), 1999.

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

[8] 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.

[9] K. Calvert, M. Doar, E. Zegura, "Modeling Internet Topology," *IEEE Communications*, 35(6), pp. 160-163, 1997.

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

[11] W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Trans. on Computers*, C-18(10), pp. 885-889, 1969.

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

[13] L. Dowdy and D. Foster, "Comparative Models of the File Assignment problem," *ACM Computing Surveys*, 14(2), pp. 287-313, 1982.

[14] K. Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing Letters*, pp. 304-307, 1974.

[15] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Trans. Networking*, 9(4), pp. 253-285, 2001.

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

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

[18] 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.

[19] 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.

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

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

[22] S. Khan and I. Ahmad, "Heuristic-based Replication Schemas for Fast Information Retrevial over the Internet," in *Proc. of 17th International Conference on Parallel and Distributed Computing Systems, San Fransisco*, U.S.A., 2004, pp. 278-283.

[23] S. Khan and I. Ahmad, "A Powerful Direct Mechanism for Optimal WWW Content Replication," in *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, U.S.A., 2005, p. 86.

[24] Y. 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*, 14(7), pp. 1332-1348, 1996.

[25] B. Li, M. Golin, G. Italiano and X. Deng, "On the Optimal Placement of Web Proxies in the Internet," in *Proc. of the IEEE INFOCOM*, 2000, pp. 1282-1290.

[26] T. Loukopoulos, and I. Ahmad, "Static and Adaptive Distributed Data Replication using Genetic Algorithms," *Journal of Parallel and Distributed Computing*, 64(11), pp. 1270-1285, 2004.

[27] R. MacAfee and J. McMillan, "Auctions and Bidding," *Journal of Economic Literature*, vol. 25, pp. 699-738, 1987.

[28] S. Mahmoud and J. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. on Database Systems*, 1(1), pp. 66-78, 1976.

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

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

[31] M. Rabinovich, "Issues in Web Content Replication," *Data Engineering Bulletin*, 21(4), pp. 21-29, 1998.

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

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

[34] G. Zipf, Human Behavior and the Principle of Least-Effort, Addison-Wesley, 1949.