Data Replication in Large Distributed Computing Systems using Discriminatory Game Theoretic Mechanism Design

Samee Ullah Khan and Ishfaq Ahmad

Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX-76019, USA
{sakhan, iahmad}@cse.uta.edu

Abstract. 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 user access time and 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. 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.

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 [4].

The Internet can be considered as a large-scale distributed computing system [11]. 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 [17]. 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 [7].

This paper uses the concepts of game theory to formally specify a mechanism with selfish agents. 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 [11].

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

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 [11], [13] and [15]). 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.

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 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:

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

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

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

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

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

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

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

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:

"Identify entries of 0/1 in the X matrix that minimizes $C_{overall}$, subject to the storage capacity constraint:

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

and subject to the primary copies policy:

$X_{p_k k} = 1 \;\; \forall \left(1 \le k \le N\right)$ *."*

In the generalized case, the DRP is NP-complete [13].

3 The Mechanism

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.

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 [17].

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)* [7].

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. *Nash equilibrium only depends upon subgame strategy profiles played along the equilibrium path* [7].■

Theorem 1. *In Nash equilibrium each player's repeated game (supergame) strategy need only be optimal along the equilibrium path* [17].■

3.3 Mechanism Applied to 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:
   - *Form:* A discriminatory mechanism.
   - *Valuation:* Valuations that are obtainable via the system parameters.
   - *Information:* Independent of any other subgame.
4. Establish the fact that subgames conform to Nash equilibrium provided agents play optimally.
5. Use Lemma 1 to establish that the entire game at instance t is in Nash equilibrium.
6. Use Theorem 1 to establish that the entire supergame is in Nash equilibrium.

Supergame: A supergame $\Gamma(\infty)$ is defined as a mechanism that is played infinite 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.

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.

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*.

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. 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.

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:

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

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.

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, 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.

Subgame Nash equilibrium: To understand the bidding behavior in a discriminatory mechanism, we shall, for simplicity, assume that the agents are ex-ante symmetric, *i.e.*, the agents are able to calculate in advance of the resolution of uncertainty. That is, we shall suppose that for all bidders $k = 1,\ldots, N, f_k(v) = f(v)$ for all $v \in [0,1]$. It is to noted 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:$[0,1]\rightarrow\mathbf{R}_+$, mapping each of his values into a bid.

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:

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

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

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

Looking closely at the left-hand side of (8), we see that is just the derivative of the product $F^{N-1}(v)\ \hat{b}(v)$ with respect to v. With this observation, we can rewrite (8) as:

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

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

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

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:

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

which can be written as:

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

There are two things to notice about the bidding function in (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:

$$\hat{b}(v) = \frac{1}{v^{N-1}}\int_0^v x\,dx^{N-1} \tag{13}$$

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

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

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

$$= \left(\frac{N-1}{N}\right)v\,. \tag{14}$$

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:* $\hat{b}(v) = (N\text{-}1/N)v$*, then the game at instance t and the supergame is in Nash equilibrium.*
Proof. It follows from Lemma 1 and Theorem 1 and Theorem 2.■

We are now ready to present the pseudo-code (Fig. 1) 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

```
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 the 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      $B_k$ = 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 available space and termination conditions*/
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 player list */
20   END WHILE
21  ENDPARFOR
22 END WHILE
```

Fig. 1. Mechanism game at instance *t*

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.

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 $O(NlogN)$. Hence, we conclude that the worst case running time of the mechanism is $O(N^2 logN)$.■

4 Experimental Setup and 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. Details of various topologies can be obtained from [11].

To evaluate the chosen replication placement techniques on realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 website [15]. Each experimental setup was evaluated thirteen times, *i.e.*, the Friday (24 hours) logs from May 1, 1998 to July 24, 1998. 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 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 [10]). For fine-grained replication, the algorithms proposed in [11], [13], and [15] are the only ones that address the problem domain similar to ours. We select from [15] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches; thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [9] (Dutch (DA) and English auctions (EA)) and [13] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons. Due to space limitations we do not give particulars 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. Notice that the discriminatory mechanism has an acronym of MECH.

4.2 Comparative Game Analysis

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 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 Fig. 2, 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.

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 Figs. 3 and 4 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 [13]). 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 Fig. 5. 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

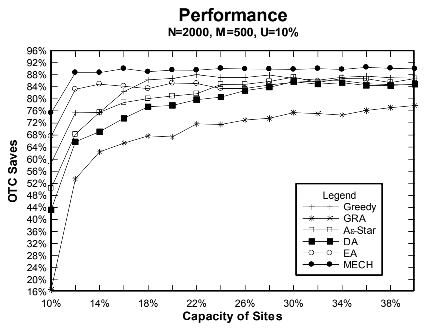

Fig. 2. OTC versus capacity of sites

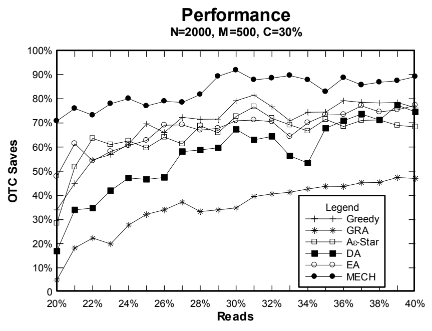

Fig. 3. OTC versus reads

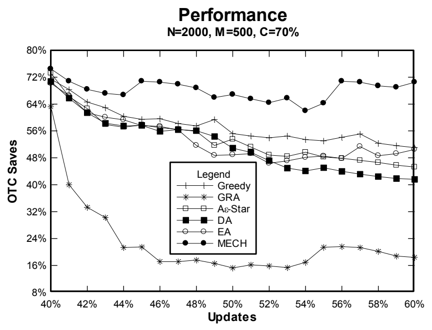

Fig. 4. OTC versus updates

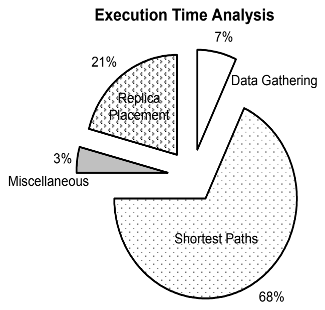

Fig. 5. Execution time analysis

depicted in Table 1.

Various problem instances were recorded with *C*=20%, 35% and *U*=25%, 35%. 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 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 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

Table 1. Running time in second [C=35%, U=35%]

Problem instance	Greedy	GRA	Aε-Star	DA	EA	MECH
M=300, N=1450	206.26	326.82	279.45	**95.64**	178.90	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	127.10	191.24	**124.73**
M=300, N=2000	275.63	469.38	368.89	143.94	197.93	**142.16**
M=400, N=1450	321.60	492.10	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**	221.10	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	402.20	660.86	460.44	**246.43**	284.63	259.56
M=500, N=1550	478.10	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 2. Average OTC (%) savings [C=35%, U=35%]

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

context of solution quality, but Aε-Star, EA and DA do indeed give a good competition, with a savings within 5%-10% of MECH.

5 Related Work

The data replication problem (see Sec. 2 for a formal description) is an extension of the classical file allocation problem (FAP). Chu [4] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [3] extended this work by distinguishing between updates and read file requests. Eswaran [6] proved that Casey's formulation was NP complete. In [14] 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 [5]. Apers in [1] considered the data allocation problem (DAP) in distributed databases where the query execution strategy influences allocation decisions. In [12] the authors proposed several algorithms to solve the data allocation problem in distributed multimedia databases (without replication), also called as video allocation problem.

Most of the research papers outlined in [5] 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 [2] 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 [18] 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. In [16], 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.

6 Conclusion

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

1. Apers, P.: Data allocation in distributed database systems. ACM Trans. Database Systems **31** (1988) 263-304
2. Awerbuch, B., Bartal, Y., Fiat, A.: Competitive distributed file allocation. In: 25th ACM

Symposium on Theory of Computation. (1993) 164-173
3. Casey, R.: Allocation of copies of a file in an information network. In: Spring Joint Computer Conference. (1972) 617-625
4. Chu, W.: Optimal file allocation in a multiple computer system. ACM Trans. on Database Systems **C-18** (1969) 885-889
5. Dowdy, L., Foster, D.: Comparative models of the file assignment problem. ACM Compt. Surveys **14** (1982) 287-313
6. Eswaran, K.: Placement of records in a file and file allocation in a computer network. Information Processing Letters **1** (1974) 304-307
7. Green, J., Laffont, J.: Characterization of satisfactory mechanisms for the revelation of preferences for public goods. Econometrica **45** (1977) 427-438
8. Kangasharju, J., Roberts, J., Ross, K.: Object replication strategies in content distribution networks. In: Web Caching and Content Distribution Workshop. (2001) 455-466
9. Khan, S. Ahmad, I.: Web content replication: A solution from game theory. Technical Report CSE-2004-05, University of Texas at Arlington (2004)
10. Khan, S., Ahmad, I.: Heuristic-based replication schemas for fast information retrieval over the internet. In: 17th International Conference on Parallel and Distributed Computing Systems. (2004) 278-283
11. Khan, S., Ahmad, I.: A powerful direct mechanism for optimal www content replication. In: 19th IEEE International Parallel and Distributed Processing Symposium. (2005)
12. Kwok, Y., Karlapalem, K., Ahmad, I., Pun, N.: Design and evaluation of data allocation algorithms for distributed database systems. IEEE J. on Selected areas in Comm. **14** (1996) 1332-1348
13. Loukopoulos, T., Ahmad, I.: Static and adaptive distributed data replication using genetic algorithms. Journal of Parallel and Distributed Computing **64** (2004) 1270-1285
14. Mahmoud, S., Riordon, J.: Optimal allocation of resources in distributed information networks. ACM Trans. on Database Systems **1** (1976) 66-78
15. Qiu, L., Padmanabhan, V., Voelker, G.: On the placement of web server replicas. In: IEEE INFOCOM. (2001) 1587-1596
16. Rabinovich, M.: Issues in web content replication. Data Engineering Bull. **21** (1998) 21-29
17. Vickrey, W.: Counter-speculations, auctions, and competitive sealed-bid tenders. J. of Finance **16** (1961) 8-37
18. Wolfson, O., Jajodia, S., Hang, Y.: An adaptive data replication algorithm. ACM Trans. on Database Systems **16** (1997) 255-314
19. Zipf, G.: Human Behavior and the Principle of Least-Effort. Addison-Wesley (1949)