A Cooperative Game Theoretical Replica Placement Technique

Samee Ullah Khan and Ishfaq Ahmad
Electrical and Computer Engineering Computer Science and Engineering
Colorado State University University of Texas at Arlington
samee.khan@colostate.edu *iahmad@cse.uta.edu*

Abstract

Creating replicas of frequently accessed data objects across a read intensive network can lead to reduced communication cost and end-user response time. On the contrary, data replication in the presence of writes incurs extra cost due to multiple updates. The selection of data objects and servers requires solving a constraint optimization problem, which is NP-complete in general. A majority of the current state-of-the-art replica placement techniques suffer from high computational complexity issues. To circumvent such issues, we propose a cooperative game theoretical technique in which the servers (players) in the system collectively deliberate to converge at a replica schema that is beneficial to the system as a whole. In particular we make use of the Aumann-Shapley mechanism of cooperative game theory to propose an effective replica placement technique that yields good solutions when the system is very heavily loaded. Experimental comparisons are made against: 1) branch and bound, 2) greedy, 3) genetic, 4) Dutch auction, and 5) English auction. As demonstrated by the experimental results, the proposed technique maintains superior solution quality in terms of lower communication cost and reduced execution time.

1. Introduction

Data replication across a read intensive network can potentially reduce the network traffic, which, in turn, can lower the response times experienced by end-users. On the other hand, with rapid updates, maintaining a large number of replicas can incur a prohibitively high overhead [19]. Therefore, efficient and effective replica schemas strongly depend on how many replicas to be placed in the system, but more importantly where.

A number of replica placement techniques for large distributed computing systems have been proposed with the underlying assumption that the servers cooperate with one another in order to layout a replica schema that optimizes the overall system performance. For instance, almost all content distribution networks (CDNs) related replica placement techniques (e.g. [6], [11], [12], [23]) rely on a centralized decision making body which optimizes a given objective (e.g. to reduce the communication cost) regardless of the costs incurred by each server [13]. These previously reported techniques are plausible as they advance the study of replica placements, however, they are very tedious and have very high computational complexity [8]. For instance, some techniques require that the underlying infrastructure be a tree [12], and the best possible bound (reported in [17]) is of the order of $O(M^3N^2)$, where M is the number of servers and N is the number of (data) objects, respectively, in the system.

To study the cooperative behavior of the servers and to derive a scalable replica placement technique, we make use of game theoretical techniques. Each server in the system plays a cooperative replica placement game (COOP). In COOP each server has two possible actions for each object. If an access is made to an object that is located at a nearby server, then the server is better off redirecting the request to that server. On the other hand if the object is located at a far off server, then the server is better off replicating that object. These decisions by the servers are not taken individually but collectively.

The goal of this paper is to see whether these servers in COOP, can layout replica schemas that converge to global optimum solution(s) targeted towards reducing the communication cost induced by accessing the objects.

A cooperative game is defined as a game in which players can conclude a binding agreement as to what outcome will be chosen to exploit the possibility of common interests. Cooperation in the sense of game theory does not mean that players sacrifice their interests for the sake of others, only that each communicates and coordinates its actions for the purpose of furthering their interests. Due to the fact that servers in a large distributed computing system can share resources, all of them should cooperate to obtain the best possible benefit.

In this paper, the Aumann-Shapley resource allocation mechanism of cooperative game theory will be used for the replica placement problem. The proposed methodology not only ensures that the total communication cost is globally minimized, but also that the data allocation is fair leading to load balancing.

The COOP technique is experimentally compared against: 1) branch and bound [14], 2) greedy [23] and 3) genetic algorithm [19] using GT-ITM [25] and Inet [6] network topology generators and Soccer World Cup 1998 traffic logs [2]. The experimental results reveal that COOP maintains superior solution quality with reduced execution time.

The remainder of this paper is organized as follows. In Section 2 we present a formal description of the replica placement problem. Section 3 focuses on describing the cooperative game theoretical replica placement technique. The experimental results, related work, and concluding remarks are provided in Sections 4, 5, and 6, respectively.

2. The Replica Placement Problem

Consider a large distributed system comprising M servers. Let S_i and s_i be the name and the total storage capacity, respectively, of server i where $1 \le i \le M$. The M servers of the system are connected by a communication network. A link between two servers 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 servers S_i and S_j. If the two servers 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 server S_i to the server S_j. Without the loss of generality we assume that $c(i,j) = c(j,i)$. Let there be N objects, each identifiable by a unique name O_k and size in simple data unites o_k where $1 \le k \le N$. Let r_{ik} and w_{ik} be the total number of reads and writes, respectively, initiated from S_i for O_k.

Let P_k, be the server 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 server of the k-th object. Each primary server P_k, contains information about the whole replication scheme R_k of O_k. This can be done by maintaining a list of the servers where the k-th object is replicated at. Moreover, every server S_i stores a two-field record for each object. The first field is its primary server P_k and the second the nearest neighborhood server NN_{ik} of server S_i which holds a replica of object k. It is possible that $NN_{ik} = S_i$, if S_i holds the replica or is the primary server of O_k. Another possibility is that $NN_{ik} = P_k$, if the primary server is the closest one holding a replica of O_k. When a server 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 server can update every object. Updates of an object O_k are performed by sending the updates to its primary server P_k, which afterwards broadcasts it to every server in R_k.

For the replica placement problem, we are interested in minimizing the total communication cost (TCC) due to object movement. There are two components affecting TCC. The first component is due to the read requests. Let R_{ik} denote the read cost, due to S_is' reading requests for object O_k, addressed to the nearest server NN_{ik}. This cost is given by:

$$R_{ik} = r_{ik} o_k c\left(i, NN_{ik}\right), \tag{1}$$

where $NN_{ik} = \{Server\ j \mid j \in R_k \,\&\&\, \min\ c(i,j)\}$. The second component is the cost arising due to the writes. Let W_{ik} be the write cost, due to S_is' writing requests for object O_k, addressed to the primary server P_k. This cost is given by:

$$W_{ik} = w_{ik} o_k \left(c\left(i, P_k\right) + \sum_{\forall j \in R_k, j \ne i} c\left(P_k, j\right) \right). \tag{2}$$

The cumulative TCC is given by:

$$\sum_{i=1}^{M} \sum_{k=1}^{N} \left(R_{ik} + W_{ik}\right). \tag{3}$$

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, with boolean elements. Using the above formulation, the replica placement problem (RPP) can be defined as:

$$(\mathbf{RPP}) \qquad Z = \min \sum_{i=1}^{M} \sum_{k=1}^{N} \left(R_{ik} + W_{ik}\right) x_{ik} \tag{4}$$

such that

$$x_{ik} \in \{0,1\}, \forall i, \forall j, (1 \le i \le M), (1 \le k \le N) \tag{5}$$

$$\prod_{i=1}^{M} x_{ik} = 1, \forall k, (1 \le k \le N) \tag{6}$$

$$\sum_{k=1}^{N} o_k x_{ik} \le s^i, \forall i, \left(1 \le i \le M\right) \tag{7}$$

$$x_{P_k k} = 1, \forall P_k, \forall k, (1 \le k \le N) \tag{8}$$

Constraint (5) is the mapping constraint, when $x_{ik} = 1$ an object O_k is held at server S_i and 0 otherwise. Constraint (6) is the participation constraint on the servers, such that all servers need to deliberate for the replica schema for all the objects in the system. Constraint (7) relates to the fulfillment of the memory space restriction, and constraint (8) tells us that no primary object can be relocated.

3. Cooperative Game Theoretical Replica Placement Game

3.1. The Aumann-Shapley Mechanism

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

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

sharing as a cooperative game, namely, the Aumann-Shapely mechanism [3]:

$$\psi_i(f, x`) = \int_0^1 \frac{\partial f(tx`)}{\partial x_i} dt , \quad (9)$$

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

Game theoreticians have proven that the Aumann-Shapley mechanism generates a unique allocation that is continuous [5], aggregate invariant [21], fully distributive [1], and satisfies the fairness principle [24]. We will see that its application to the data replication problem will not generate a mismatch of resource allocation which is the one of the primary reasons for obtaining sub-optimal solutions.

3.2. Replica Placement Game (COOP)

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

3.2.1. Aumann-Shapley Replica Placement Game

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

$$f(Z) = \min \sum_{i=1}^{M} \sum_{k=1}^{N} (R_{ik} + W_{ik}) x_{ik} , \quad (10)$$

subject to the constraints (5), (6), (7), and (8). The Aumann-Shapley mechanism, ψ, at server i, will be given as:

$$\psi_i = \int_0^1 \frac{\partial f(tZ)}{\partial Z} dt , \quad (11)$$

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

$$\sum_{i=1}^{M} Z\psi_i = f(Z) . \quad (12)$$

Now, each sever would incur a communication cost equal to $Z\psi_i$ due to the accesses made to the data objects hosted by that server. In order to bring a meaning to the Aumann-Shapley mechanism, we need to solve RPP in conjunction to the Aumann-Shapley mechanism. This can be done very efficiently by taking the Lagrangian of RPP. However, the Lagrangian function on RPP in conjunction with the Aumann-Shapley mechanism using non-linear programming methods would generate multiple solutions, which is not what we desire. (A detailed discussion on the instability of the Aumann-Shapely mechanism involving non-linear programming methods can be found in [4].) To negate the problem of multiple solutions, we take the Lagrangian on the dual of RPP, Z_D, in conjunction with the Aumann-Shapley mechanism, $Z_D(\psi)$, and we get:

$$Z_D(\psi) = \begin{array}{l} \min \sum_{i=1}^{M} \sum_{k=1}^{N} (R_{ik} + W_{ik}) x_{ik} \\ + \sum_{i=1}^{M} \psi_i \left(\sum_{k=1}^{N} o_k x_{ik} - s_i \right) \end{array} , \quad (13)$$

subject to constraints (5), (6), and (8).

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

$$Z_D(\psi) = \begin{array}{l} \min \sum_{k=1}^{N} \left(\sum_{i=1}^{M} ((R_{ik} + W_{ik}) + \psi_i o_k) x_{ik} \right) \\ - \sum_{i=1}^{M} \psi_i s_i \end{array} . \quad (14)$$

Then the Lagrangian dual problem is as follows:

$$Z_{LD} = \max_{\psi \ge 0} (Z_D(\psi)) . \quad (15)$$

For a fixed, ψ, (13) can be decomposed into sub-problems each of which corresponds to individual server's communication cost as illustrated in (12). Each sub-problem is a bounded variable knapsack problem. (This fact is inline with the initial proof of NP-hardness of the data placement problem, where the authors in [6] showed a reduction to the binary knapsack problem.) These sub-problems can easily be solved by a dynamic programming algorithm with a running time $O(M^3N^2)$ [17]. However, the algorithm works only when the lower bound on every variable is 0. This is certainly not the case with RPP and as discussed in Section 3.1 the Aumann-Shapley mechanism has continuous first partial derivatives; therefore, the lower bounds on some variables may be positive. For this purpose, we need to devise a technique that can cater for the possible positive lower bounds and which is original to this research.

To find Z_{LD}, we need to find a ψ which gives a maximum of $Z_D(\psi)$ over all $\psi_i \ge 0$. For this purpose we make use of the celebrated sub-gradient method [10] coupled with a branch-and-bound technique to prune and refine the sub-gradient method. Now suppose we are given a current ψ^t at iteration t and an optimal replica placement x_{ik}^t to $Z_D(\psi^t)$, the next step is decided by:

$$\psi^{t+1} = \max \left\{ 0, \psi^t + \alpha^t \left(\sum_{k=1}^{N} o_k x_{ik} - s_i \right) \right\} , \quad (16)$$

where $\alpha^t = \left(Z^* - Z_D(\psi^t) \right) \Big/ \sum_{i=1}^{M} \left(\sum_{k=1}^{N} o_k x_{ik} - s_i \right)^2$, (17)

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

The COOP Technique

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

INC(i)

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

DEC(i)

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

Figure 1: The pseudo-code for the COOP procedure.

After a specified number of iterations, (a)

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

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

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

Note that we have used criterion (b) instead of $Z_D(\psi^l) \geq Z^*$ since the Lagrangian generated costs are integral. Criteria (c) and (d) represent the optimality conditions. The whole process of the COOP technique is presented in Figure 1.

Now suppose that case (b) does not occur within the iteration limit (50 in our implementation). If the current solution satisfies (c) we have found a new feasible solution. Then we update Z^* and continue the sub-gradient iterations. Suppose (d) as well as (c) occur, *i.e.*, the optimality conditions hold within the iteration limit. If the current node is the root node of the branch-and-bound tree, the algorithm ends with the solution x_{ik}, an optimal solution to RPP. Otherwise, Z^* is updated and we continue the sub-gradient iterations. On the other hand, if the optimality conditions do not hold within the iteration limit, we branch at that node and generate two or more child nodes to further improve on the result. We set $\psi^0 = 0$ at the root node and use the Lagrangian at the parent node as the initial value at the child node to avoid unnecessary computations at the child node as suggested in [5]. The selection of the next node to solve is based on the best bound rule.

3.2.2. Feasibility of the COOP Technique

It rarely happens that the solution to the Lagrangian dual problem is feasible to the original problem. However, it can often be transformed to a feasible solution by a minor modification. A solution to $Z_D(\psi)$ satisfies (5), (6), and (8) but may violate (7). So we modify the solution so that it satisfies (7). Let us define:

$$\partial = \left\{ \forall i, (1 \leq i \leq M) \mid \sum_{k=1}^{N} o_k x_{ik}^* - s_i < 0 \right\}, \quad (18)$$

$$\sigma = \left\{ \forall i, (1 \leq i \leq M) \mid \sum_{k=1}^{N} o_k x_{ik}^* - s_i \geq 0 \right\}, \quad (19)$$

where x^*_{ik} is the current infeasible replica placement. Note that ∂ and σ are the set of indices of constraints (7) which are violated by the solution. If we decrease $\sum o_k x^*_{ik}$ for $i \in$

σ, we may be able to make the solution feasible to the problem represented by the current node. Any decrease of the solution values does not affect the validity of constraints (7), but a careless decrease may cause the solution to violate some of the constraints (6) and/or (8). On the other hand, if we increase $\sum o_k x^*_{ik}$ for $i \in \partial$, before we decrease $\sum o_k x^*_{ik}$ for $i \in \sigma$, then it is more likely that the modified solution becomes feasible. So the increment and decrement of solution should be determined carefully.

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

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

3.2.3. Branching Rules of the COOP Technique

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

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

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

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

4. Experimental Results

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory. COOP was implemented using Ada and Ada GNAT's distributed systems annex GLADE [22].

To establish diversity in our experimental setups, the network connectively was changed considerably. We used GT-ITM [2] for the network topologies, the procedure for which is as follows: A random graph $G(M,P(\text{edge} = p))$ with $0 \le p \le 1$ contains all graphs with nodes (servers) M in which the edges are chosen independently and with a probability p. The pure random topologies were obtained with p = {0.4, 0.5, 0.6, 0.7, 0.8}. In each of these topologies the distance between two serves was reversed mapped to the communication cost of transmitting a 1kB of data and the latency on a link was assumed to be 2.8×10^{-8} m/s (copper wire).

To evaluate the replica allocation methods under realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 web server [2]. Each experimental setup was evaluated thirteen times, *i.e.*, only the Friday (24 hours) logs from May 1, 1998 to July 24, 1998. (The Friday logs have the heaviest traffic compared to any other day of the week.) To process the logs, we wrote a script that returned: only those objects which were present in all the logs (25,000 in our case), the total number of requests from a particular client for an object, the average and the variance of the object size. From this log we chose the top five hundred clients (maximum experimental setup). A random mapping was then performed of the clients to the nodes of the topologies. Note that this mapping is not 1-1, rather 1-M. This gave us enough skewed workload to mimic real world scenarios. It is also worthwhile to mention that the total amount of requests entertained for each problem instance was in the range of 1-2 million. The primary replicas' original server was mimicked by choosing random locations. The capacities of the servers $C\%$ were generated randomly with range from *Total Primary Object Sizes/2* to *1.5×Total Primary Object Sizes*. The variance in the object size collected from the access logs helped to instill enough miscellanies to benchmark object updates. The updates were randomly pushed onto different servers, and the total system update load was measured in terms of the percentage update requests $U\%$ compared that to the initial network with no updates.

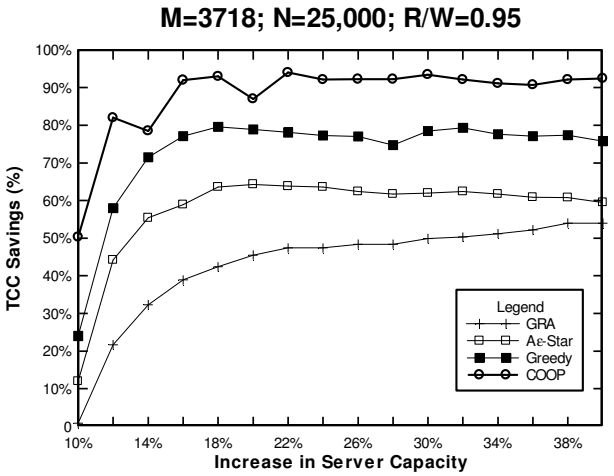

Figure 2: TCC savings versus capacity.

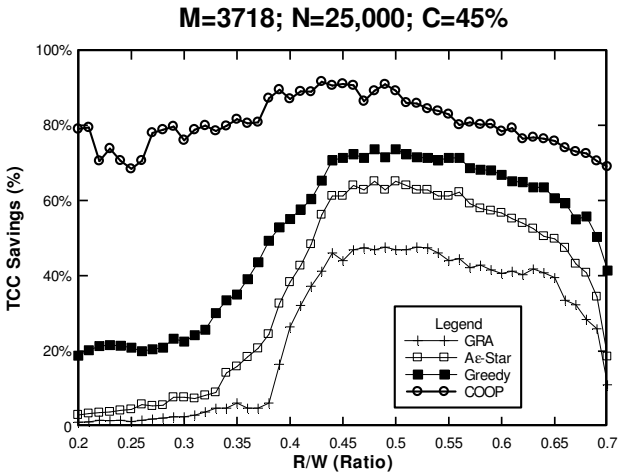

Figure 3: TCC savings versus read/write ratio.

Table 1: Running time of the replica placement methods in seconds.

(a): Small problem instances [C=20%, R/W=0.45]				
Problem Size	**Greedy**	**GRA**	**Aε-Star**	**COOP**
M=200, N=500	84.13	111.19	116.61	**64.59**
M=200, N=1000	91.90	115.68	123.56	**58.30**
M=200, N=1500	93.91	121.21	136.62	**60.87**
M=300, N=500	114.28	152.30	168.93	118.14
M=300, N=1000	131.00	150.04	178.59	134.61
M=300, N=1500	162.25	178.30	215.68	196.75
M=400, N=500	151.68	184.95	238.52	**149.92**
M=400, N=1000	161.58	202.17	284.00	196.81
M=400, N=1500	169.29	245.31	324.75	175.65

(b): Large problem instances [C=45%, R/W=0.85]				
Problem Size	**Greedy**	**GRA**	**Aε-Star**	**COOP**
M=2500, N=15,000	310.14	491.00	399.63	**211.64**
M=2500, N=20,000	330.75	563.25	442.66	339.12
M=2500, N=25,000	357.74	570.02	465.52	370.38
M=3000, N=15,000	452.22	671.68	494.60	556.98
M=3000, N=20,000	467.65	726.75	498.66	**341.61**
M=3000, N=25,000	469.86	791.26	537.56	549.38
M=3718, N=15,000	613.27	883.71	753.87	742.70
M=3718, N=20,000	630.39	904.20	774.31	**629.67**
M=3718, N=25,000	646.98	932.38	882.43	654.33

Since the access logs are of the year 1998, we first use Inet [6] topology generator to estimate the number of nodes in the network. This number came up to be 3718, *i.e.*, there were 3718 AS-level nodes in the Internet at the time when the Soccer World Cup 1998 was being played. Therefore, we set the upper bound on the number of servers in the system at M = 3718.

Comparative algorithms: For comparison, we chose three types of replica allocation methods. To provide a fair comparison, the assumptions and system parameters were kept the same in all the methods. For the data replication problem, the techniques proposed in [14], [17], [19] and [23] are the only ones that address the problem domain similar to ours. We select from [23] the greedy approach (Greedy) for comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [17]); thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [14] (the efficient branch and bound based technique Aε-Star) and [19] (the genetic algorithm based method GRA) are also among the chosen techniques for comparisons. We encourage the readers to obtain an insight on the comparative techniques from the referenced papers.

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

Comparative analysis: We observe the effects of increase in storage capacity. An increase in the storage capacity means that a large number of objects can be replicated. Replicating an object that is already extensively replicated, is unlikely to result in significant traffic savings as only a small portion of the servers will be affected overall. Moreover, since objects are not equally read intensive, increase in the storage capacity would have a great impact at the beginning (initial increase in capacity), but has little effect after a certain point, where the most beneficial ones are already replicated. This is observable in Figure 2, which shows the performance of the algorithms. GRA performed the worst. COOP and Greedy showed an immediate initial increase (the point after which further replicating objects

is inefficient) in its TCC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most TCC savings (57%) followed by Greedy with 44%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this paper due to space restrictions) that the increase in capacity from 10% to 19%, resulted in 4.7 times (on average) more replicas for all the algorithms.

Next, we observe the effects of increase in the read and write frequencies. Since these two parameters are complementary to each other, we describe them together. To observe the system utilization with varying read/write frequencies, we kept the number of servers and objects constant. Increase in the number of reads in the system would mean that there is a need to replicate as many object as possible (closer to the users). However, the increase in the number of updates in the system requires the replicas be placed as close as to the primary server as possible (to reduce the update broadcast). This phenomenon is also interrelated with the system capacity, as the update ratio sets an upper bound on the possible traffic reduction through replication. Thus, if we consider a system with unlimited capacity, the "replicate everywhere anything" policy is strictly inadequate. The read and update parameters indeed help in drawing a line between good and marginal algorithms. The plot in Figure 3 shows the results of read/write ratio against the TCC savings. A clear classification can be made between the algorithms. COOP and Greedy incorporate the increase in the number of reads by replicating more objects and thus savings increased up to 92%, while GRA gained the least of the TCC savings of up to 42%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial selection of gene population (for details see [19]). Moreover, GRA maintains a localized network perception. Increase in updates result in objects having decreased local significance (unless the vicinity is in close proximity to the primary location). On the other hand, COOP, Aε-Star and Greedy never tend to deviate from their global view of the problem.

Lastly, we compare the termination time of the algorithms. Various problem instances were recorded with C = 20%, 45% and R/W = 0.45, 0.85. The entries in Table 1 made bold represent the fastest time recorded over the problem instance. It is observable that Greedy terminated faster than all the other techniques, followed by COOP, Aε-Star, and GRA.

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

5. Related Work

The replica placement problem is an extension of the classical file allocation problem (FAP). Chu [7] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [5] extended this work by distinguishing between updates and read file requests. Eswaran [9] proved that Casey's formulation was NP-complete. In [20] Mahmoud *et al.* provide an iterative approach that achieves good solution quality when solving the FAP for infinite server capacities.

Recently, game theory has emerged as a popular tool to tackle optimization problems especially in the field of distributed computing. However, in the context of data replication it has not received much attention. We are aware of only three major works which directly or indirectly deal with the data replication problem using game theoretical techniques. The first work [8] is mainly on caching and uses an empirical model to derive Nash equilibrium. The second work [15] focuses on mechanism design issues and derives an incentive compatible auction for replicating data on the Web. The third work [16] deals with identifying Nash strategies derived from synthetic utility functions. Our work differs from all the game theoretical techniques in: 1) identifying a cooperative non-priced based replica allocation method to tackle the data replication problem, 2) using game theoretical technique to study an environment that closely mimics the Internet, and 3) thoroughly comparing the proposed technique against several conventional replica placement techniques.

Readers are encouraged to see [18] for a comprehensive survey on the conventional replica placement techniques.

6. Conclusions

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

The proposed COOP technique improved the performance relative to other conventional methods in four ways. First, the number of replicas in a system was controlled to reflect the ratio of read versus write access. To maintain concurrency control, when an object is updated, all of its replicas need to be updated simultaneously. If the write access rate is high, there should be few replicas to reduce the update overhead. If the read access rate is overwhelming, there should be a

high number of replicas to satisfy local accesses. Second, performance was improved by replicating objects to the servers based on locality of reference. This increases the probability that requests can be satisfied either locally or within a desirable amount of time from a neighboring server. Third, replica allocations were made in a fast algorithmic turn-around time.

As future work, we would extend the current system model to incorporate regional or hierarchical mechanisms. This would enable the system to be less vulnerable to the failures of a single mechanism, and in turn would open the realms of devising hierarchical games, where in each level either a cooperative or non-cooperative game could be played to replicate data.

7. References

[1] M. Aoki, *The Co-operative Game Theory of the Firm*, Oxford University Press, New York, 1984.

[2] 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.

[3] R. J. Aumann and L. S. Shapley, *Values of Non-Atomic Games*, Princeton University Press, 1974.

[4] J. Bracken and G. McCormick, *Selected Application of non-linear Programming*, Wiley, New York, 1980.

[5] D. Campbell, *Resource Allocation Mechanisms*, Cambridge University Press, 1987.

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

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

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

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

[10] M. L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Sciences*, vol. 27, pp. 1-18, 1981.

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

[12] 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.

[13] M. Karlsson and M. Mahalingam, "Do We Need Replica Placement Algorithms in Content Delivery Networks?" in *Proc. of Web Caching and Content Distribution Workshop*, 2002, pp. 117-128.

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

[15] 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*, 2005.

[16] N. Laoutaris, O. Telelis, V. Zissimopoulos and I. Stavrakakis, "Local Utility Aware Content Replication," in *IFIP Networking Conference*, 2005.

[17] 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.

[18] T. Loukopoulos, D. Papadias and I. Ahmad, "An Overview of Data Replication on the Internet," in *Proc. of International Symposium on Parallel Architectures, Algorithms and Networks*, 2002, pp. 31-38.

[19] 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.

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

[21] A. Mas-Collel, W. Whinston and J. Green, *Microeconomic Theory*, Oxford University Press, 1995.

[22] L. Pautet and S. Tardieu, "GLADE: A Framework for Building Large Object-Oriented Real-Time Distributed Systems," in *3rd International Symposium on Object-Oriented Real-Time Distributed Systems*, 2000, pp. 244-251.

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

[24] H. P. Young, "Cost Allocation," Chapter 34, *Handbook of Game Theory*, vol. 2, R. J. Aumann and S. Hart, eds., Elsevier Science B. V., 1994.

[25] E. Zegura, K. Calvert and M. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Trans. On Networking*, 5(6), pp. 770-783, 1997.