N+1st Price Auction Based Replica Schemas

Samee Ullah Khan and Ishfaq Ahmad

***Abstract*—**This paper addresses the problem of fine-grained data replication on a set of Internet sites using the N+1st price auction. Specifically, we present an adaptive auction mechanism for replication of objects in a distributed system. The mechanism is adaptive in the sense that it changes the replica schema of the objects by continuously moving the schema towards an optimal one, while ensuring object concurrency control. The mechanism is experimentally evaluated against three well-known techniques from the literature: greedy, branch and bound, and genetic algorithms. The experimental results reveal that the proposed approach outperforms the three techniques in both the execution time and solution quality.

***Keywords*—**Data replication, auctions, static allocation, pricing.

I. Introduction

FINE-GRAINED (object based) replica schemas determine how many replicas of each objects are created, and to which sites they are assigned. These schemas critically affect the performance of the distributed computing system (e.g. the Internet), since reading an object locally is less costly than reading it remotely [9]. Therefore, in a read intensive network an extensive replica schema is required. On the other hand, an update of an object is written to all, and therefore, in a write intensive network a constricted replica schema is required. In essence replica schemas are strongly dependent upon the read and write patterns for each object [3]. Recently, a few approaches on replicating data objects over the Internet have been proposed in [1], [6], [7], [8] and [10]. The majority of the work related to data replication on the Internet employs the coarse-grained (site based) replication. As the Internet grows and the limitations of caching become more obvious, the importance of fine-grained replication, *i.e.*, duplicating highly popular data objects, is likely to increase [8].

In this paper, the replica schemas are established in a static fashion. The aim is to identify a replica schema that effectively minimizes the object transfer cost. We propose a novel technique based on the N+1st price auction, where the players compete for memory space at sites so that replicas can be placed. This approach is compared against three well-known techniques from the literature: branch and bound [7], greedy [10], and genetic algorithms [8]. Experimental results reveal that this simple and intuitive approach outperforms the three techniques in both execution time and solution quality.

Samee Ullah Khan is with the Department of Computer Science and Engineering, University of Texas at Arlington, TX 76019 USA (phone: 817-272-3607; fax: 817-272-3784; e-mail: sakhan@cse.uta.edu).

Ishfaq Ahmad is with the Department of Computer Science and Engineering, University of Texas at Arlington, TX 76019 USA (e-mail: iahmad@cse.uta.edu).

TABLE I
NOTATIONS AND THEIR MEANINGS

Symbol	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.

II. Problem formulation

Consider a distributed system comprising M sites, with each site having its own processing power, memory (primary storage) and media (secondary storage). Let S_i and s_i be the name and the total storage capacity (in simple data units e.g. blocks), respectively, of site i where $1 \leq i \leq M$. The M sites of the system are connected by a communication network. A link between two sites S_i and S_j (if it exists) has a positive integer $c(i,j)$ associated with it, giving the communication cost for transferring a data unit between sites S_i and S_j. If the two sites are not directly connected by a communication link then the above cost is given by the sum of the costs of all the links in a chosen path from site S_i to the site S_j. Without the loss of generality we assume that $c(i,j) = c(j,i)$. This is a common assumption (e.g. see [7], [8], and [10]). 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. Our replication policy assumes the existence of one primary copy for each object in the network. Let P_k, be the site which holds the primary copy of O_k, *i.e.*, the only copy in the network that cannot be de-allocated, hence referred to as primary site of the k-th object. Each primary site P_k, contains information about the whole replication scheme R_k of O_k. This can be done by maintaining a list of the sites where the k-th object is replicated at, called from now on the *replicators* of O_k. Moreover, every site S_i stores a two-field record for each object. The first field is its primary site P_k and the second the nearest neighborhood site NN_k^i of site S_i which holds a replica of object k. In other words, NN_k^i is the site for which the reads from S_i for O_k, if served there, would incur the minimum possible communication cost. It is possible that $NN_k^i = S_i$, if S_i is a *replicator* or the primary site of O_k. Another possibility is that

$NN_k^i = P_k$, if the primary site is the closest one holding a replica of O_k. When a site S_i reads an object, it does so by addressing the request to the corresponding NN_k^i. For the updates we assume that every site can update every object. Updates of an object O_k are performed by sending the updated version to its primary site P_k, which afterwards broadcasts it to every site in its replication scheme R_k.

For the DRP under consideration, we are interested in minimizing the total Replication Cost (RC) (or the total network transfer cost) due to object movement, since the communication cost of control messages has minor impact to the overall performance of the system. There are two components affecting RC. The first component of RC is due to the read requests. Let R_k^i denote the total RC, due to S_is' reading requests for object O_k, addressed to the nearest site NN_k^i. This cost is given by the following equation:

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

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

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

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

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

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

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

Sites which are not the *replicators* of object O_k create RC equal to the communication cost of their reads from the nearest *replicator*, plus that of sending their writes to the primary site of O_k . Sites belonging to the replication scheme of O_k, are associated with the cost of sending/receiving all the updated versions of it. Using the above formulation, the Data Replication Problem (DRP) can be defined as: *Find the assignment of* 0,1 *values in the X matrix that minimizes* $C_{overall}$, *subject to the storage capacity constraint*: $\sum_{k=1}^{N} X_{ik} o_k \leq s_i \forall (1 \leq i \leq M)$, and *subject to the primary copies policy*: $X_{P_k k} = 1 \forall (1 \leq k \leq N)$.

In the generalized case, DRP is essentially a constraint optimization problem, reducible to the *Knapsack* problem, and without the storage constraint, to the *minimum k-median* problem [8].

III. N+1ST PRICE AUCTION AND ITS APPLICATION

In the auction setup each primary copy of an object k is a player. A player k can perform the necessary computations on its strategy set by using the site P_k's (where it resides) processor. At each given instance a (sub)-auction takes place at a particular site i chosen in a round robin fashion from the set of M sites. These auctions are performed continuously throughout the system's life, making it a self evolving and self repairing system. However, for simulation purposes ("cold" network [8]) we discrete the continuum solely for the reason to observe the solution quality.

Each player k competes through bidding for memory at a site i. Many would argue that memory constraints are no longer important due to the reduced costs of memory chips. However, replicated objects (just as cached objects) reside in the memory (primary storage) and not in the media (secondary storage) [9]. Thus, there will always be a need to give priority to objects that have higher access (read and write) demands. Moreover, memory space regardless of being primary or secondary is limited.

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

$$B_k^i = R_k^i - (\sum_{x=1}^{M} w_k^x o_k C(i, SP_k) - W_k^i) \tag{5}$$

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

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

FIGURE I
PSEUDO-CODE FOR N+1ST PRICE AUCTION MECHANISM

code for the N+1st price auction is given in Figure I.

To present our algorithm, 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. The auction mechanism performs in steps. In each step a server S_i is chosen from LS in a round-robin fashion. Each player $k \in O$

calculates the benefit function of object. The set O represents the collection of players that are legible for participation. A player k is legible if and only if the benefit function value obtained for site S_i is the maximum of among all the other benefit function values for sites other than i, *i.e.*, $S_i \geq S_{-i}$. This is done in order to suppress mediocre bids, which, in turn improves computational complexity. It is to be noted that in each step L^i together with the corresponding nearest server value NN_k^i, are updated accordingly.

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

IV. Experimental results and discussions

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

TABLE II
OVERVIEW OF TOPOLOGIES

Topology	**Mathematical Representation**
SGRG [7] (12 topologies)	Randomized layout with node degree (d*) and Euclidian distance (d) between nodes as parameters.
GT-ITM PR [2] (5 topologies)	Randomized layout with edges added between the randomly located vertices with a probability (p).
GT-ITM W [2] (9 topologies)	$P(u,v)=\alpha e^{-d/(\beta L)}$
SGFCGUD [7] (5 topologies)	Fully connected graph with uniform link distances (d).
SGFCGRD [7] (5 topologies)	Fully connected graph with random link distances (d).
SGRGLND [7] (9 topologies)	Random layout with link distance having a lognormal distribution [4].

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. We used existing topology generator toolkits and also self generated networks. Table II summarizes the various techniques used to gather forty-five various topologies. All the results reported, represent the average performance over all the topologies.

To evaluate our proposed technique on realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 website [2]. Each experimental setup was evaluated thirteen times, *i.e.*, 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 choose the top five hundred clients (maximum experimental setup), which were randomly mapped to one of the nodes of the topologies. 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 instill 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.

For comparison, we selected three various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. We chose 1) from [7] the efficient branch-and-bound based technique (Aε-Star), 2) from [8] the genetic algorithm based technique (GRA) which showed excellent adaptability against skewed workload, 3) from [10] the famous greedy approach (Greedy). Due to space limitations, details for a specific technique are left as a reading exercise.

TABLE III
RUNNING TIME IN SECONDS

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

Table III (best times shown in bold) shows the algorithm execution times. The number of sites was kept constant at 500, and the number of objects was varied from 1350 to 2000.

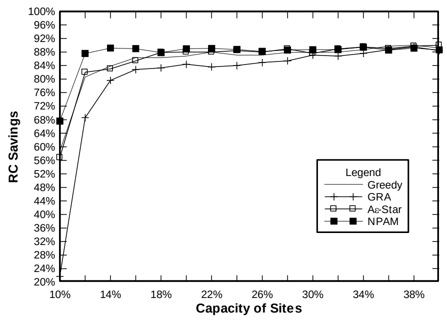

FIGURE II
RC SAVINGS VERSUS SYSTEM CAPACITY (N=2000, M=500, U=5%)

With maximum load (2000 objects and 500 sites), the proposed technique NPAM saved approximately 50 seconds of termination time then the second fastest algorithm (Greedy).

Superiority of execution time comes at the cost of loss in solution quality. However, NPAM showed high solution quality. First, we observe the effects of system capacity increase. An increase in the storage capacity means that a large number of objects can be replicated. Replicating an object that is already extensively replicated, is 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 II, which shows the performance of the algorithms. Greedy and NPAM showed an immediate initial increase (the point after which further replicating objects is inefficient) in its RC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most RC savings (35%) followed by Greedy with 29%. Further experiments with 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 17%, resulted in 4 times (on average) more replicas for all the algorithms.

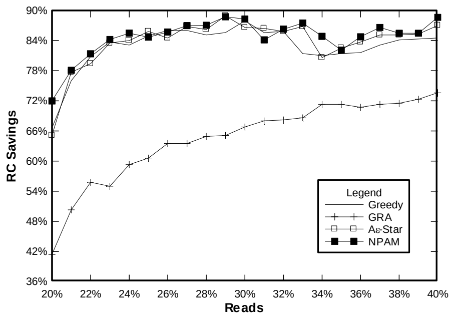

FIGURE III
RC SAVINGS VERSUS READS (N=2000, M=500, C=45%)

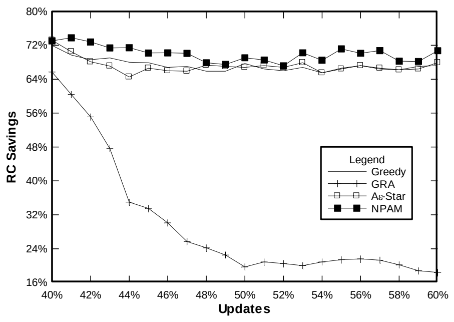

FIGURE IV
RC SAVINGS VERSUS UPDATES (N=2000, M=500, C=60%)

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 III and IV show the results of read and update frequencies, respectively. A clear classification can be made between the algorithms. Aε-Star, Greedy and NPAM incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 89%. GRA gained the least of the RC savings of up to 67%. To understand why there is such a gap in the performance between the algorithms, we recall from [8] that GRA specifically depends on the initial population of the candidate solution. Moreover, GRA maintains a localized network perception. Increase in updates result in objects having decreased local significance (unless the vicinity is in close proximity to the primary location). On the other hand, Aε-Star, Greedy and NPAM never tend to deviate from their global view of the problem domain.

TABLE IV
AVERAGE RC SAVINGS IN PERCENTAGE

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

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

V. CONCLUSIONS

This paper proposed an N+1st price auction mechanism that effectively addressed the fine-grained data replication problem. The experimental results which were recorded against some well-known techniques such as branch and bound, greedy, and genetic algorithms revealed that the proposed mechanism exhibited 5%-10% better solution quality and incurred the fastest execution time.

REFERENCES

[1] T. Abdelzaher and N. Bhatti, "Web content adaptation to improve sever workload behavior," *Computer Networks*, 21(11), pp. 1536-1577, 1999.

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

[3] R. Bunt, D. Eager, G. Oster, and C. Williamson, "Achieving load balance and effective caching in clustered web servers," in *4th International Web Caching Workshop*, pp. 159-169, 1999.

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

[5] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking*, 9(4), pp. 253-285, 2001.

[6] J. Kangasharju, J. Roberts and K. Ross, "Object replication strategies in content distribution networks," in *Proc. of WCCD*, pp. 455-466, 2001.

[7] 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*, 2004.

[8] T. Loukopoulos and I. Ahmad, "Static and adaptive data replication algorithms for fast information access in large distributed systems," in *Proc. of ICDCS*, pp. 385-392, 2000.

[9] T. Loukopoulos, D. Papadias, and I. Ahmad, "An overview of data replication on the internet," in *Proc. of ISPAN*, pp. 31-36, 2002.

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