RAMM: A Game Theoretical Replica Allocation and Management Mechanism

Samee Ullah Khan and Ishfaq Ahmad
Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX-76019, U.S.A.
{sakhan, iahmad}@cse.uta.edu

Abstract

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

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

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

The remainder of this paper is organized as follows. Section 2 formulates the data replication problem (DRP). Section 3 describes the RAMM. The experimental results, related work and concluding remarks are provided in Sections 4, 5 and 6, respectively.

2. 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. 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.

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:

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

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 by the following equation:

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

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

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

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

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

Using the above formulation, the DRP can be defined as:

Find the assignment of 0, 1 values in the X matrix that minimizes $C_{overall}$, subject to the storage capacity constraint: $\sum_{k=1}^{N} X_{ik} o_k \leq s^i \forall (1 \leq i \leq M)$, *and subject to the primary copies policy*: $X_{P_k k} = 1 \quad \forall (1 \leq k \leq N)$.

3. The RAMM

The Basics: The mechanism contains M agents. Each agent i has some private data $t^i \in \mathbf{R}$. This data is termed as the agent's *true data* or *true type*. Only agent i has knowledge of t^i. Everything else in the mechanism is public knowledge. Let t denote the vector of all the true types $t = (t^1 \ldots t^M)$.

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

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

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

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

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

Bids: Each agent i is interested in reporting a bid b^i such that it maximizes its profit, regardless of what the other agents bid, *i.e.*, $u^i(t^i, (b^{-i}, t^i)) \geq u^i(t^i, (b^{-i}, b^i))$ for all b^{-i} and b^i. It is to be noted that truth telling ($b^i = t^i$) brings in more utility to the agents than over bidding and under bidding

For more details on the optimality of such type of payment and bidding procedures see [15]. In that paper, the authors have identified many such scenarios, all but reporting truthfully fail to exploit this (second best) payment option.

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

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

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

Below we detail these two objectives.

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

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

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

To capture the effect of i's report on the outcome, we introduce a hypothetical *null report*, which corresponds to agent i reporting that it is indifferent among the possible decisions and cares only about payments. When i makes the null report, the mechanism optimally chooses the decision $D(X,M\text{-}i,t^{-i})$. The resulting total value of the decision for the set of agents M-i would be $V(X,M\text{-}i,t^{-i})$, and the mechanism might also provide an agent i with payment equivalent to $h^i(t^{-i})$. Thus, if i makes a null report, the total payoff to the agents in set M-i is $V(X,M\text{-}i,t^{-i}) + h^i(t^{-i})$. This would mean that the RAMM would choose payments for the M-i agents regardless of what i reports to the RAMM. For a detailed analysis of the above payment structure, readers are encouraged to see [10]. It is to be noted that in economic game theoretical literature this type of payment is often referred to as Vickrey payments [10].

We have entertained all the pending optimization issues regarding the RAMM, and are ready to give a pseudo-code (Figure 1).

Briefly, we maintain a list L^i at each server. This list contains all the objects that can be replicated by agent i onto site S^i. We can obtain this list by examining the two constraints of the DRP. List L^i would contain all the objects that have their size less then the total available space b^i. Moreover, if site S^i is the primary host of some object k', then k' should not be in L^i. We also maintain a list LS containing all sites that can replicate an object, *i.e.*, $S^i \in LS$ if $L^i \neq$NULL. The algorithm works iteratively. In each step the mechanism asks all the agents to send their preferences (first **PARFOR** loop). Each agent i recursively calculates the true data of every object in list L^i. Each agent then reports the dominant true data (line 08) to the mechanism. The mechanism receives all the corresponding entries, and then chooses the best dominant true data. This is broadcasted to all the agents, so that they can update their nearest neighbor table NN_k^i, which is shown in Line 20 (NN^i_{OMAX}). The object is replicated and payments made to the agent. The mechanism progresses forward till there are no more agents interested in acquiring any data for replication.

The above discussion allows us to deduce the

The RAMM Algorithm

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

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

Figure 1: Pseudo-code describing the RAMM.

following result about the RAMM algorithm.

Theorem 1: *RAMM takes $O(MN^2)$ time.*

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

4. Experiments

We performed experiments on a 440MHz Ultra 10 machine with 512MB memory. The experimental evaluations were targeted to benchmark the placement policies. The RAMM was implemented using IBM Pthreads. The solution quality is measured in terms of network communication cost (OTC percentage) that is saved under the replication scheme found by the algorithms, compared to the initial one, *i.e.*, when only primary copies exists.

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 GT-ITM [3] topology generator to obtain 45 random network topologies. In all the topologies the distance of the link between nodes was equivalent to the communication cost. To evaluate the chosen replication placement techniques 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.*, the Friday (24 hours) logs from May 1, 1998 to July 24, 1998. Thus, each

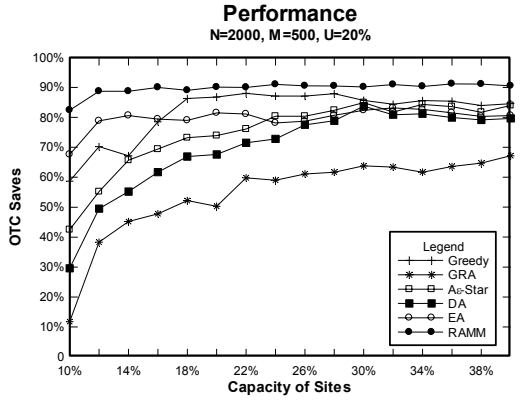

Figure 2 OTC savings versus capacity.

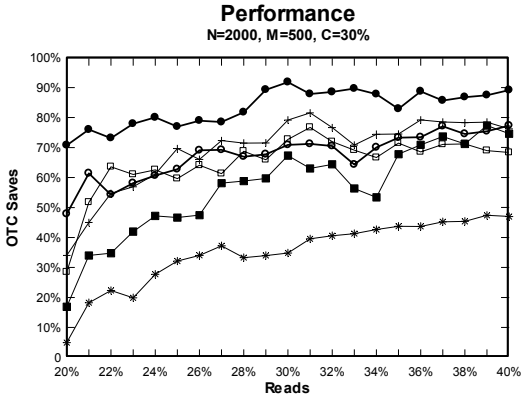

Figure 3 OTC savings versus reads.

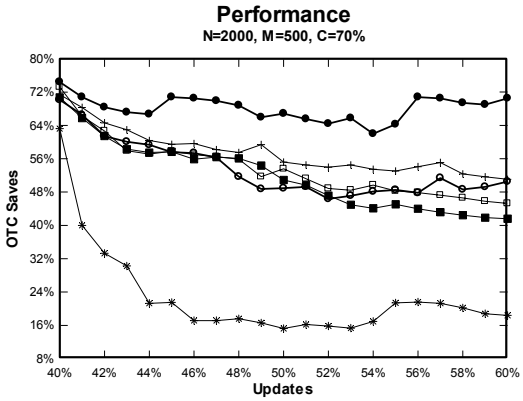

Figure 4 OTC savings versus updates.

Table 1 Running time (sec.) [*C*=55%, *U*=10%].

Problem Size	Greedy	GRA	Aε-Star	DA	EA	RAMM
M=300, *N*=1400	206.26	326.82	279.45	**95.64**	178.9	97.98
M=300, *N*=1450	236.61	379.01	310.12	115.19	185.15	**113.65**
M=300, *N*=1500	258.45	409.17	333.03	127.1	191.24	**124.73**
M=300, *N*=1550	275.63	469.38	368.89	**143.94**	197.93	147.16
M=300, *N*=2000	298.12	475.02	387.94	**158.45**	204.29	159.12
M=400, *N*=1400	348.53	536.96	368.03	**187.26**	223.56	195.41
M=400, *N*=1450	366.38	541.12	396.96	**192.41**	221.1	214.55
M=400, *N*=1500	376.85	559.74	412.17	**208.92**	245.47	218.73
M=400, *N*=1550	389.71	605.63	415.55	**215.24**	269.31	223.92
M=400, *N*=2000	391.55	659.39	447.97	**224.18**	274.24	235.17
M=500, *N*=1400	478.1	689.44	511.69	**257.96**	301.72	266.42
M=500, *N*=1450	485.34	705.07	582.71	**269.45**	315.13	272.68
M=500, *N*=1500	511.06	736.43	628.23	**278.15**	324.26	291.83
M=500, *N*=1550	525.33	753.5	645.26	**289.64**	331.57	304.47
M=500, *N*=2000	539.15	776.99	735.36	**312.68**	345.94	317.6

Table 2 Running time (sec.) [*C*=15%, *U*=55%].

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

Table 3 Average OTC (%) savings.

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

experimental setup in fact represents an average of the 585 (13×45) data set points. To process the logs, we wrote a script that returned: only those objects which were present in all the logs (2000 in our case), the total number of requests from a particular client for an object, the average and the variance of the object size. From this log we chose the top five hundred clients (maximum experimental setup). A random mapping was then performed of the clients to the nodes of the topologies. Note that this mapping is not 1-1, rather 1-*M*. This gave us enough skewed workload to mimic real world scenarios. It is also worthwhile to mention that the total amount of requests entertained for each problem instance was in the range of 1-2 million. The primary replicas' original site was mimicked by choosing random locations. The capacities of the sites *C*% were generated randomly with range from *Total Primary Object Sizes/2* to *1.5×Total Primary Object 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.

4.1. Comparative Algorithms

For comparison, we selected five various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. The techniques studied include efficient branch-and-bound based technique (Aε-Star [9]). For fine-grained replication, the algorithms proposed in [10], [11], [12], and [14] are the only ones that address the problem domain similar to ours. We select from [14] the greedy approach (Greedy) for

comparison because it is shown to be the best compared with 4 other approaches (including the proposed technique in [11]); thus, we indirectly compare with 4 additional approaches as well. Algorithms reported in [10] (Dutch (DA) and English auctions (EA)) and [12] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons. Due to space limitations details for a specific technique are left as an exercise for the readers and can be obtained from the referenced papers.

4.2. Comparative 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 Figure 2, which shows the performance of the algorithms. GRA once again performed the worst. The gap between all other approaches was reduced to within 12% of each other. DA and RAMM showed an immediate initial increase (the point after which further replicating objects. is inefficient) in its OTC savings, but afterward showed a near constant performance. GRA although performed the worst, but observably gained the most OTC savings (47%) followed by Greedy with 44%. Further experiments with various update ratios (5%, 10%, and 20%) showed similar plot trends. It is also noteworthy (plots not shown in this paper due to space restrictions) that the increase in capacity from 10% to 17%, resulted in 3.7 times (on average) more replicas for all the algorithms.

Next, we observe the effects of increase in the read and update (write) frequencies. Since these two parameters are complementary to each other, we describe them together. In both the setups the number of sites and objects were kept constant. Increase in the number of reads in the system would mean that there is a need to replicate as many object as possible (closer to the users). However, the increase in the number of updates in the system requires the replicas be placed as close as to the primary site as possible (to reduce the update broadcast). This phenomenon is also interrelated with the system capacity, as the update ratio sets an upper bound on the possible traffic reduction through replication. Thus, if we consider a system with unlimited capacity, the "replicate everywhere anything" policy is strictly inadequate. The read and update parameters indeed help in drawing a line between good and marginal algorithms. The plots in Figures 3 and 4 show the results of read and update frequencies, respectively. A clear classification can be made between the algorithms. Aε-Star, EA, Greedy and RAMM incorporate the increase in the number of reads by replicating more objects and thus savings increase up to 86%. GRA gained the least of the OTC savings of up to 13%. To understand why there is such a gap in the performance between the algorithms, we should recall that GRA specifically depends on the initial population (for details see [12]). Moreover, GRA maintains a localized network perception. Increase in updates result in objects having decreased local significance (unless the vicinity is in close proximity to the primary location). On the other hand, Aε-Star, EA, Greedy and RAMM never tend to deviate from their global view of the problem search space.

Lastly, we compare the termination time of the algorithms depicted in Tables 1 and 2. Various problem instances were recorded with C=15%, 55% and U=10%, 55%. The entries in bold represent the fastest time recorded over the problem instance. It is observable that RAMM and DA terminated faster than all the other techniques, followed by EA, Greedy, Aε-Star and GRA.

Table 3 shows the quality of the solution in terms of OTC percentage for eight problem instances (randomly chosen), each being a combination of various numbers of sites and objects, with varying storage capacity and update ratio. For each row, the best result is indicated in bold. The proposed RAMM algorithm steals the show in the context of solution quality, but Aε-Star, EA and DA do indeed give a good competition, with a savings within a range of 5%-10% of RAMM.

5. Related Work

The data replication problem as presented in Section 2 is an extension of the classical file allocation problem (FAP). Chu [5] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [4] extended this work by distinguishing between updates and read file requests. Eswaran [7] proved that Casey's formulation was NP complete. In [13] 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 [6]. Apers in [1] considered the data allocation problem (DAP) in distributed databases where the query execution strategy influences allocation decisions.

Most of the research papers outlined in [6] 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*. In [16] Wolfson *et al.* proposed an algorithm that leads to optimal single file replication in the case of a tree network. The performance of the scheme for general network topologies is not clear though. Dynamic replication protocols were also considered under the Internet environment. Heddaya *et al.* [8] proposed protocols that load balance the workload among replicas.

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

6. Conclusions

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

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

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

References

[1] P. Apers, "Data Allocation in Distributed Database Systems," *ACM Trans. Database Systems*, 13(3), pp. 263-304, 1988.

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

[4] R. Casey, "Allocation of Copies of a File in an Information Network," in *Proc. of IFIPS Congress*, 1972, pp. 617-625.

[5] W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Trans. on Computers*, 18(10), pp. 29-33, 1969.

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

[7] K. Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," in *Proc. of IFIPS Congress*, 1974, pp. 304-307.

[8] A. Heddaya and S. Mirdad, "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents," *in Proc. 17th IEEE ICDCS*, 1997, pp. 160-168.

[9] S. Khan and I. Ahmad, "Heuristic-based Replication Schemas for Fast Information Retrevial over the Internet," in *Proc. of 17th PDCS*, 2004, pp. 278-283.

[10] S. Khan and I. Ahmad, "A Powerful Direct Mechanism for Optimal WWW Content Replication," To appear in *Proc. of the 19th IEEE IPDPS*, 2005, p. 86.

[11] 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*, 1999, pp. 1282-1290.

[12] 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.

[13] S. Mahmoud and J. Riordon, "Optimal allocation of resources in distributed information networks," *ACM Trans. on Database Systems*, 1(1), pp. 66-78, 1976.

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

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

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