Distrib Parallel Databases (2010) 28: 187–218
DOI 10.1007/s10619-010-7069-0

Replicating data objects in large distributed database systems: an axiomatic game theoretic mechanism design approach

Samee Ullah Khan · Ishfaq Ahmad

Published online: 29 September 2010

Abstract Data object replication onto distributed servers can potentially alleviate bottlenecks, reduce network traffic, increase scalability, add robustness, and decrease user perceived access time. The decision of selecting data object and server pairs requires solving a constraint optimization problem that in general is NP-complete. In this paper, we abstract the distributed database system as an agent-based model, wherein agents continuously compete for allocation and reallocation of data objects. Each agent aims to replicate objects onto its server such that the communication cost is minimized. However, these agents do not have a global view of the system. Thereby, the optimization process becomes highly localized. Such localized optimization may severely affect the overall system performance. To cope with such localized optimization, we propose a "semi-distributed" axiomatic game theoretical mechanism. The mechanism's control is unique in its decision making process, wherein all the heavy processing is done on the servers of the distributed system and the central body is only required to take a binary decision: (0) not to replicate or (1) to replicate. The cost model used by the agents in the mechanism for the purpose of identifying beneficial data objects is tailored made so that even though the agents take decisions based on their local knowledge domain, the effect is translated into a system-wide performance enhancement. The proposed mechanism is extensively compared against seven well-known conventional and three game theoretical replica allocation methods, namely, branch and bound, greedy, genetic, data-aware replication, tree inspired bottom-up procedure, tree inspired min-max procedure, Benders'

Communicated by Ashfaq Khokhar.

S.U. Khan
Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA
e-mail: samee.khan@ndsu.edu

I. Ahmad (✉)
Department of Computer Science and Engineering, University of Texas, Arlington, TX, USA
e-mail: iahmad@cse.uta.edu

decomposition based procedure, game theoretical English auction, game theoretical Dutch auction, and game theoretical selfish replication procedure. The experimental setup incorporates GT-ITM, Inet network topology generators, Soccer World Cup 1998 access logs, and NASA Kennedy Space Center access logs to closely mimic the Web in its infrastructure and user access patterns. The experimental results reveal that the proposed technique despite its non-cooperative nature improves the solution quality and reduces the execution time compared to other techniques.

Keywords Algorithmic mechanism design · Game theory · Optimization · Replication · Distributed algorithms · Placements

1 Introduction

Distributed database systems store data onto a number of geographically located servers. These systems provide a much better solution for applications, such as banking, than monolithic centralized systems. Newer technologies in the domain of mobile computing are aiding in the openness, scalability, and flexibility of distributed databases. In essence, a distributed database system facilitates access to shared data objects. As distributed database systems grown in size, data replication plays an increasing important role. Data replication increases data availability in the presence of server or communication failures and decreases retrieval costs by local access (if possible). The maintenance of replicated data is closely related to the communication cost. Data replication management can have significant impact on the overall system performance. Data replication management in distributed database systems concerns the decision of when and where to allocate physical copies of logical data fragments (replica placement), and when and how to update them to maintain an acceptable degree of consistency (write control). Discussions in [23, 28, 42, 45, 54] reveal that client(s) experience reduced access latencies provided that data is replicated within their close proximity. However, this is applicable in cases when only read accesses are considered. If updates of the contents are also under focus, then the locations of the replicas must be: (a) in close proximity to the client(s) and (b) in close proximity to the primary copy. For fault-tolerant and highly dependable systems, replication is essential, as demonstrated in a real world example of OceanStore [26]. Therefore, efficient and effective replica allocation methods strongly depend on how many replicas to be placed in the system, and more importantly where [30].

Numerous methods have been proposed that address the data object replication problem in large-scale distributed database systems. (For a recent survey see [44].) However, all of the previous methods operated under the generic assumption that the servers cooperated with each other to replicate data objects that are beneficial to the system as a whole. For instance, in a content distribution network (CDN) the *distribution system* moves contents to the replica servers [20, 23, 27]. This distribution system acts as a centralized decision making body, and makes decisions on where and what to replicate by observing the system's parameters, such as server load, storage capacity, and communication latencies. Although distributed systems encourage distributed control, yet they require that the overall system performance criteria be

met. Managing a large-scale distributed database system through a single entity (distribution system) requires huge amounts of data processing in order to find a feasible replica schema and is vulnerable to system failures [9]. This being the motivation, in this paper, we propose a "semi-distributed" [1] (a hybrid of both the centralized and decentralized approaches) replication technique, by which all the heavy processing is done on the servers of the distributed system and the central body is only required to take a binary decision: (0) not to replicate or (1) to replicate. This leads to a simple yet an efficient scheme with enhanced scalability and low complexity that grows in proportion to the number of servers in the system.

For the proposed semi-distributed replica allocation technique, we abstract the distributed database system as an agent-based model wherein agents continuously compete for allocation and reallocation of data objects. An agent is a computational entity that is capable of autonomous behaviour in the sense of being aware of the options available to it when faced with a decision making task related to its domain of interest [30]. In such a competitive model, there is no *a priori* motivation for cooperation. Moreover, the agents due to their localized view of the problem domain are encouraged to take decisions individually that are beneficial only to themselves. To cope with such an individualism and localized optimization, new mechanisms based on novel oracles need to be derived. The objective of the mechanisms should be to allow the agents to take decisions that are based on their local information, which essentially translate into the overall system performance enhancement.

Briefly, the proposed game theoretical mechanism works as follows. It first asks all the agents to provide a list of objects that are beneficial for replication onto their servers. Having obtained this data, the mechanism makes the allocation and informs the agents. For every allocation the mechanism makes a payment to each agent (to compensate for hosting object(s)). Each agent's goal is to provide the mechanism with a list of objects that maximize its benefit. The mechanism on the other hand, is designed such that, the agents maximize their benefit only if the reported list contains objects that when replicated, brings the overall system-wide communication cost to the minimum. Thus, the agents which are competing against each other in a non-cooperative game collaborate unknowingly to optimize the overall system-wide objective.

This paper aims to formally specify a semi-distributed axiomatic game theoretical replica allocation mechanism that clearly defines the necessary and sufficient conditions that must be met to develop a heuristic that can achieve superior performance with reduced computational complexity. For this purpose, we leverage upon our prior work reported in [30–34] to identify from each of them the necessary ingredients for a semi-distributed game theoretical procedure.

Note that there is a significant difference between the definition of truthfulness used in [32, 33] and the one used in this paper. The randomized replica allocation technique in [32, 33] yields a truthful dominant strategy for any possible random choice of the algorithm; however, the mechanism is truthful only in expectation, a weaker notion of truthfulness [52]. A randomized mechanism can be seen as a probability distribution over deterministic mechanisms: an element x is selected randomly and the corresponding mechanism is used. So, the mechanism in [33] is truthful for every fixed x. Moreover, in [30], the notion of utility is replaced by the expected

utility one: even though the expected utility is maximized when telling the truth, for some x there might exist a better (untruthful) strategy. We distinguish ourselves from the above mentioned work by deviating in the definition of truthfulness by reporting a deterministic truthful outcome. Highlighting the fact that revenue is a major concern, [34] suggests examining auctions (or mechanisms) that do not necessarily maximize the social welfare, and characterizes all truthful mechanisms for this problem. The characterization of [34] is a special case of ours, for bounded (predefined number of replicas in the system) replica allocations, and it also appears implicitly in [30, 32, 33]. In [31], the authors also ignore the social welfare, instead they attempt to compute various functions of the agents valuations (such as the order statistics) using auctions of minimal communication complexity.

From the above discussion, we can deduce that the results reported in this paper are from game theoretical perspective significantly different from other previously reported works by the authors of this paper. The general implementation procedure for any mechanism design is the same. That is, players in a game bid for an item (data object), the mechanism selects a bid, and the allocation of resources is made. Such a fact can be deduces from all of the papers related to mechanism design based optimization procedures, such as [30–34]. Therefore, rather than reinventing procedures, in this research, our goal is to identify the necessary and sufficient conditions in the form of axioms with the purpose of: if all of the axioms are followed, then automatically a global optimal solution for any minimization problem can be achieved. Such an axiomatic approach was not present in any of the previous work [30–34]. Moreover, the claim that any minimization problem can be solved using the axiomatic procedure derived in this paper is universal and unique to this research. To show the effectiveness of the axiomatic procedure we solve the data replication problem that was originally reported in [45]. The data replication problem requires minimization of the overall system-wide communication cost and several extensions of the original problem formulation are possible. However, for the sake of consistency and comparative procedures, we do not modify our original problem formulation. Because the axiomatic procedure proposed in this research is subjected to a problem that has been previously tackled by other game theoretical procedures, the implementation of the axiomatic procedure converges to similar derivations of previously reported procedures. However, the previous procedures are considerable difficult and cumbersome to derive for a given problem. Therefore, this is another forte of the proposed axiomatic procedure.

The major contributions of this paper are as follows:

1. We derive a general purpose axiomatic game theoretical mechanism. This mechanism ensures that although the agents use self-beneficial strategies, yet the effect is translated into a global optimization.
2. The essence of this mechanism is captured in six well-defined axioms that exhibit properties of global optimality, truthfulness and utilitarianism.
3. We use these axioms to obtain an efficient algorithm for the data replication problem.
4. The proposed semi-distributed replica allocation mechanism is experimentally compared with some well-known techniques, such as branch and bound [28],

greedy [54], genetic [45], data-aware replication [8], tree inspired bottom-up procedure [43], tree inspired min-max procedure [43], Benders' decomposition based procedure [4], game theoretical English auction [28], game theoretical Dutch auction [28], and game theoretical selfish replication procedure [41] using GT-ITM [61] and Inet [10] network topology generators, and Soccer World Cup 1998 traffic logs [1] and NASA Kennedy Space Center traffic logs [50]. The experimental results reveal that the proposed technique's solution quality is competitive to other centralized approaches and exhibits fast execution time.

The remainder of this paper is organized as follows. In Sect. 2 we summarize some important related work followed by a formal description of the data replication problem in Sect. 3. Section 4 focuses on describing the generalized axiomatic game theoretical mechanism along with its properties. Section 5 concentrates on modeling the mechanism for the data replication problem. The experimental results and concluding remarks are provided in Sects. 6 and 7, respectively.

2 Related work

The data replication problem (see Sect. 3 for a formal description) is an extension of the classical file allocation problem (FAP). Chu [11] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [7] extended this work by distinguishing between updates and read file requests. Eswaran [16] proved that Casey's formulation was NP-complete. In [46] 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 [15].

In the context of the distributed database systems, replication algorithms fall into the following three categories: (1) the problem definition does not cater for the client accesses, (2) the problem definition only accounts for read access and (3) the problem definition considers both read and write access including consistency requirements. These categories are further classified into four categories according to whether a problem definition takes into account single or multiple objects, and whether it considers storage costs. Table 1 shows the categorized outline of the previous work reported.

The main drawback of the problem definition in category 1 is that they place the replicas of every object, in the same node. Clearly, this is not practical, when many objects are placed in the system. However, they are useful as a substitute of the problem definition of category 2, if the objects are accessed uniformly by all the clients in the system and utilization of all nodes in the system is not a requirement. In this case category 1 algorithms can be orders of magnitude faster than the ones for category 2, because the placement is decided once and it applies to all objects.

Most of the research papers tackle the problem definition of category 2. They are applicable to read-only and read-mostly workloads. In particular this category fits well in the context of CDNs. Problem definitions [20, 23, 27] have all been used in CDNs. The two main differences between them are whether they consider single or multiple objects, and whether they consider storage costs or not. The cost function

Table 1 Summary of related work. References in bold represent work that had experimental evaluations. References in quotes represent work which is evaluated and compared with work reported in this article

Category	Number of objects	Storage constraints	References
Category 1: No object access	Single object	No storage constraint	**[23]**
		Storage constraint	–
		No storage constraint	[20], **[24]**
	Multiple objects	Storage constraint	–
Category 2: Read accesses only	Single object	No storage constraint	[15], [20], **[24]**, [27], [36], [39], "[42]", "**[54]**"
		Storage constraint	[13], **[26]**, [36], **[40]**
		No storage constraint	**[25]**, [36], **[41]**, [58]
	Multiple objects	Storage constraint	[13], "**[43]**", [49]
Category 3: Read and write accesses	Single object	No storage constraint	[11], [14], **[60]**
		Storage constraint	[9], [25], [37]
		No storage constraint	[22], [39], [55], **[56]**
	Multiple objects	Storage constraint	[3], "[4]", "**[8]**", **[12]**, "**[28]**", [30], **[39]**, "**[41]**", "**[45]**", [46], this article

in [46] also captures the impact of allocating large objects and could possible be used when the object size is highly variable. In [15] the authors tackled a similar problem—the proxy cache placement problem. The performance metric used there was the distance parameter, which consisted of the distance between the client and the cache, plus the distance between the client and the node for all cache misses. It is to be noted that in CDN, the distance is measured between the cache and the closest node that has a copy of the object.

The storage constraint is important since it can be used in order to minimize the amount of changes to the previous replica placements. As far as we know only the works reported in [28] and [45] have evaluated the benefits of taking storage costs into consideration. Although there are research papers which consider storage constraints in their problem definition, yet they never evaluate this constraint (e.g. see [16, 24, 42, 54]).

Considering the impacts of writes, in addition to that of reads, is important, if content providers and applications are able to modify documents. This is the main characteristic of category 3. Some research papers in this category also incorporate consistency protocols—in many different ways. For most of them, the cost is the number of writes times the distance between the client and the closest node that has the object, plus the cost of distributing these updates to the other replicas of the object. In [23, 25, 27, 42] the updates are distributed in the system using a minimum spanning tree. In [24] and [54] one update message is sent from the writer to each copy, while in [28] and [45] a generalized update mechanism is employed. There a broadcast model is proposed in which any user can update a copy. Next, a message is sent to the primary (original) copy holder server which broadcasts it to the rest of the replicas. This approach is shown to have lower complexity than any of the above

mentioned techniques. In [39] and [56], it is not specified how updates are propagated. The other main difference among the above definitions is that [25, 27, 28, 42, 45, 58] minimize the maximum link congestion, while the rest minimize the average client access latency or other client perceived costs. Minimizing the link congestion would be useful, if bandwidth is scare.

Our work differs from all the above in: (1) describing a problem definition that combines both the server selection and replica placement problems, (2) 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", (3) indirectly incorporating the minimization of link congestion via object transfer cost, (4) extensively evaluating the impact of storage constraints similar to the evaluations performed in [28] and [45], and (5) using game theoretical techniques.

Recently, game theory has emerged as a popular tool to tackle optimization problems especially in the field of distributed computing. In the context of data replication, the authors of this paper have primarily advocated the usage of game theory for a number of years. Other works in this filed that we would like to distinguish this work from are as follows. The first work [12] is mainly on caching and uses an empirical model to derive Nash equilibrium. The second work [35] focuses on developing a cooperative solution for the data replication problem. The third work [39] deals with identifying Nash strategies derived from synthetic utility functions. Our work differs from all the game theoretical techniques in: (1) identifying a non-cooperative non-priced based replica allocation method to tackle the data replication problem, (2) using game theoretical techniques to study an environment where the agents behave in a selfish manner, (3) deriving pure Nash equilibrium and pure strategies for the agents, (4) performing extensive experimental comparisons with a number of conventional techniques using an experimental setup that is mimicking the Web in its infrastructure and access patterns.

3 Formal description of the data replication problem

The most frequently used acronyms in this paper are listed in Table 2.

Consider a large-scale distributed database system comprising M servers, with each server 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 server i where $1 \leq i \leq M$. 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 $c(i, j) = c(j, i)$. (This is a common assumption e.g. see [23, 28, 45, 54]). Let there be N (data) objects, each identifiable by a unique name O_k and size in simple data unites o_k where $1 \leq k \leq N$.

Table 2 Notations and their meanings

Symbols	Meaning
M	Total number of servers 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 server
s^i	Storage capacity of server i
r_k^i	Number of reads for object k from server i
R_k^i	Aggregate read cost of r_k^i
w_k^i	Number of writes for object k from server i
W_k^i	Aggregate write cost of w_k^i
NN_k^i	Nearest neighbor of server i holding object k
$c(i, j)$	Communication cost between servers i and j
P_k	Primary server of the k-th object
R_k	Replication schema of object k
$C_{overall}$	Total overall data transfer cost
OTC	Object transfer cost (network communication cost)
AGT-RAM	Axiomatic game theoretical replica allocation mechanism

Let r_k^i and w_k^i be the total number of reads and writes, respectively, initiated from S^i for O_k during a certain time period t. This time period t determines when to instigate the relocation period so that the replica placement algorithm can be invoked. Note that this time period t is the only parameter that requires human intervention. However, in this paper we use analytical data, which enables us to effectively predict the time interval t (see Sect. 6 for details).

Our replication policy assumes the existence of one primary copy for each object in the network. 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, called from now on the *replicators* of O_k. 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_k^i of server S^i that holds a replica of object k. That is, NN_k^i is the server 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 server of O_k. Another possibility is that $NN_k^i = P_k$, 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 updated version to its primary server P_k, which afterwards broadcasts it to every server in its replication scheme R_k. This technique of keeping data consistence is popularly known as the *lazy update* approach [15].

For the data replication problem 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 on the overall performance of the system (for an elaborate discussion on this issue and examples of possible control messages see [45]), 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.

In the data replication literature, many difference metrics are referenced, namely, (a) reducing end user access time [60], (b) data availability [16], and (c) OTC [45]. Among all of them the OTC metric is the most widely accepted, general, and effectively captures the crux of the rest of the metrics [44]. The OTC is defined as the cumulative cost of data object movement within the system to satisfy read and write requests of users connected to the servers. Minimizing the OTC would in effect: (a) reduce the end user access time because requests will be processed quickly from their closest replicas and (b) increase data availability because data has to be replicated in order for the OTC to be reduced.

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 server 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 $NN_k^i = \{\text{Server}\, j \,|\, j \in R_k \wedge \min c(i, j)\}$. 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 server P_k. This cost is given by the following equation:

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

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

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

Servers which are not the *replicators* of object O_k create OTC equal to the communication cost of their reads from the nearest *replicator*, plus that of sending their writes to the primary server of O_k. Servers belonging to the replication scheme of O_k, are associated with the cost of receiving all the updated versions of it. Using the above formulation, the data replication problem 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 $X_{P_k k} = 1 \ \forall (1 \leq k \leq N)$.

The minimization of $C_{overall}$ will have two impacts on the distributed system under consideration: First, it ensures that the object replication is done in such a way that it minimizes the maximum distance between the replicas and their respective primary copies. Second, it ensures that the maximum distance between an O_k and the user(s) accessing that object (via a server) is also minimized.

Based on the above system overview and the underlying assumptions, if we are to find an optimal placement of replicas in a large-scale distributed computing system, then we must incorporate among others the following parameters in a brute force (exhaustive search) method [30]: (a) The access frequency of each data object; (b) The time remaining until each data object is updated next; (c) The probability that each server functions properly during the lifespan of the system; (d) The probability that the network will remain connected during the lifespan of the system.

Even if some looping is possible, the computational complexity is very high, and this calculation must be done every time if any of the above parameters change. Moreover, parameters (c) and (d) cannot be formulated in practice because faults do not follow a known phenomenon. For these reasons, we take the following heuristic approach:

1. Replicas are relocated, only during the *relocation period* [21].
2. At every relocation period, replica allocation is determined based on the access (both read and update) frequency of each data object and the network topology at that moment.

In the generalized case, the data replication problem has been proven to be NP-complete [45].

4 Axiomatic game theoretical mechanism

In this section we use various building blocks to construct a generalized axiomatic game theoretical mechanism. We begin by defining a mechanism [6], which has two components: (a) the *algorithmic output*, and (b) the agents' *valuation functions*.

Definition 1 [51] A mechanism is in which:

1. The system consists of M agents. Each agent i has some private data $t^i \in \Re$. This is termed as the agent's true data or true value. Everything else in the mechanism is public knowledge.

2. The algorithmic output maps to each true data vector $t = t^1 \cdots t^M$ a set of allowed outputs $x \in X$, where $x = x^1 \cdots x^M$.
3. Each agent i's preferences are given by a real valued function $v^i(t^i, x)$ called valuation.

Remarks The valuation of an agent represents the quantification of its value from the output x, when its true data is t^i in terms of some predefined currency. For example, if the output of the mechanism is x and it hands the agent p^i amount of payment, then its utility becomes: $u^i = p^i + v^i(t^i, x)$.

We now focus on identifying a mechanism that allows the algorithmic output to optimize a given objective function. Below we give a refined definition of an optimization (minimization in our case) oriented mechanism.

Definition 2 [6] An optimization oriented mechanism is one where:

1. The algorithmic output is given by a positive real valued objective function $g(t, x)$, and
2. a set of feasible outputs X.

Thus, we require an output $x \in X$ that minimizes g, such that for any other output $x' \in X$, $g(t, x) \leq g(t, x')$. This is fine as long as we can find a mechanism that can solve a given problem by assuring that the required algorithmic output occurs, when all the agents choose strategies that maximize their utility functions (a min-max procedure). Let a^{-i} denote a vector of strategies, not including agent i, i.e, $a^{-i} = (a^1, \ldots, a^{i-1}, a^{i+1}, \ldots, a^M)$. We can define a mechanism that is able to solve a utilitarian based minimization problem as:

Definition 3 [5] A mechanism $(m = (x(\cdot), p(\cdot)))$ is composed of (a) an algorithmic output function $x(\cdot)$, and (b) the payment function $p(\cdot)$. The mechanism should have the following properties:

1. The mechanism allows for each agent i a set of strategies A^i. It is up to the agent what strategy $(a^i \in A^i)$ to adopt in order to have its utility function optimized.
2. The mechanism should provide an algorithmic output x derived from the output function, i.e., $x = x(a^i \cdots a^M)$.
3. In order to motivate the agents, the mechanism should provide a payment $p^i = p^i(a^1 \cdots a^M)$ to each of the M agents.

Remarks Recall that $x = x^1 \cdots x^M$. It is then not difficult to see that agent i's algorithmic output can easily be obtained once the mechanism identifies the output x. It is possible that agents due to their selfish nature may alter the output of the algorithm in order to fervently gather more resources. Thus, we are required to enforce the mechanism to handle the special case of selfish agents by adding the following property to Definition 3.

Property 4 The mechanism should be a representation of dominant strategies, and this is possible if for each agent i and each t^i there exists a dominant strategy

($a^i \in A^i$), such that for all possible strategies of the other agents a^{-i}, a^i maximizes agent i's utility.

Remarks Literature survey reveals that the simplest of all the mechanisms is the one where the agents' strategies are to report their true data. Such types of mechanisms are called *truthful* mechanisms, and they are based on the revelation principle [38]. This principle stresses on the fact that truthful mechanisms can be efficiently and effectively used to solve optimization problems. Moreover, it is also shown that any complicated mechanism can be reduced to the truthful mechanisms [18]. To align ourselves with Property 4 of Definition 3, we show that truth-telling is indeed a dominating strategy.

Lemma 1 *For agents to report their type* (*truth-telling*) *is a dominating strategy.*

Proof Let (a^i, a^{-i}) denote a vector of strategies of all the M agents, i.e., $(a^i, a^{-i}) = (a^1 \cdots a^M)$. Truth-telling would mean that $a^i = t^i$. Then, for every $a^{i'} \in A^i$, if we define $x = x(a^i, a^{-i})$, $x' = x(a^{i'}, a^{-i})$, $p^i = p^i(a^i, a^{-i})$, and $p^{i'} = p^{i'}(a^{i'}, a^{-i})$, then $p^i + v^i(t^i, x) \geq p^{i'} + v^i(t^i, x')$, i.e., $u^i \geq u^{i'}$. □

We can use this result to couple it with a useful theorem reported in [47]. This will characterize the mechanism as truthful. Below we state the theorem.

Theorem 1 [47] *A mechanism that implements a given problem with dominant strategies is truthful.*

In Definition 3 we stated (Property 1) that a mechanism $m = (x(\cdot), p(\cdot))$ allows the agents to maximize their utilities. Since utilities enable the agents to express their preferences, it is important to identify an oracle that does exactly that. Literature survey revealed the following result, which we append to the property list of Definition 3.

Theorem 2 [51] *A mechanism is called utilitarian if its objective function satisfies* $g(x, t) = \sum_i v^i(t^i, x)$.

Property 5 The mechanism should have an objective function that satisfies $g(x, t) = \sum_i v^i(t^i, x)$.

The above property is so useful that it can help us identify the two important components of an axiomatic game theoretical mechanism. We state:

Theorem 3 [17] *A truthful mechanism* $m = (x(t), p(t))$ *belongs to the class of minimization utilitarian mechanisms if and only if*:

1. $x(t) \in \operatorname{argmin}_x(\sum_i v^i(t^i, x))$.
2. $p^i(t) = \sum_{j \neq i} v^j(t^j, x(t)) + h^i(t^{-i})$, *where* $h^i(\cdot)$ *is an arbitrary function of* t^{-i}.

Remarks Note that conditions 1 and 2 of Theorem 3 are the exact mathematical derivations of Properties 2 and 3 of Definition 3, respectively. Moreover, the mechanism stated in Theorem 3 takes in as argument t for both the algorithmic output and

Axiomatic Game Theoretical Mechanism

Axiom 1 (Ingredients) A mechanism should have (a) an algorithmic output specification, and (b) agents' utility functions.

Axiom 2 (Agent disposition) Every agent has a private value termed as true data, everything else is public knowledge. This value along with a valuation function should reveal the preferences of the agent.

Axiom 3 (Truthful) The mechanism should have agents that project their dominant strategies.

Axiom 4 (Utilitarian) The mechanism's objective function should be to sum the agents' valuations.

Axiom 5 (Motivation) The mechanism should reward the agents with a payment. These payments are made in accordance to a specified function based on the algorithmic output.

Axiom 6 (Algorithmic output) The mechanism's algorithmic output should be a function that aids the agents to execute their preferences.

Fig. 1 Axiomatic game theoretical mechanism

the payment function, i.e., $m = (x(\cdot), p(\cdot))$ is now written as $m = (x(t), p(t))$. This is because of the mergence of Theorems 1 and 2. For convenience we shall now state the truthful mechanism in the axiomatic form (Fig. 1).

We aim to use the above (discussed) axiomatic game theoretical mechanism to find solutions for the data replication problem (Sect. 3). The six axioms defined above will act as a cast for the data replication problem. In essence, we want a replica allocation mechanism that solves the data replication problem with the properties guaranteed by the six axioms.

5 Axiomatic game theoretical replica allocation mechanism (AGT-RAM)

The directions laid down in Sect. 4 will be used to apply the axiomatic game theoretical mechanism to the data replication problem.

Ingredients (Axiom 1) The distributed system describe in Sect. 3 is considered and it consists of M agents. That is, each server is represented by an agent. Assume for the time being that the feasible output (of the algorithm exists, and) are all partitions $x = x^1 \cdots x^M$ of the objects to the agents, where x^i is the set of objects allocated to agent i. Also assume that each agent i's utility function u^i exists. (In the subsequent text it will be clear what exactly x and u^i are.)

Agent disposition (Axiom 2) An agent holds two key elements of data (a) the available server capacity b^i, and (b) the cost of replication or valuation (CoR_k^i) of object O_k to the agent's server i. There are three possible cases:

1. DRP [π]: Each agent i holds the cost to replicate $CoR_k^i = t^i$ associated with each object O_k, where as the available server capacity and everything else is public knowledge.
2. DRP [σ]: Each agent i holds the available server capacity $b^i = t^i$, where as CoR_k^i and everything else is public knowledge.

3. DRP $[\pi, \sigma]$: Each agent i holds both the cost of replication and the server capacity $\{CoR_k^i, b^i\} = t^i$, where as everything else is public knowledge.

Remarks Intuitively, if agents know the available server capacities of other agents, that gives them no advantage whatsoever. However, if they come about to know their CoR_k^i then they can modify their valuations and alter the algorithmic output. Note that an agent can only calculate CoR_k^i via the frequency of read and writes. Everything else such as the network topology, latency on communication lines, and even the server capacities can be public knowledge. Therefore, DRP$[\pi]$ is the only natural choice.

Description of valuation (CoR_k^i) We can write CoR_k^i as follows:

$$CoR_k^i = \sum_{k\in(X_k^i=1)} \left[w_i^k o_k \left(\sum_{\forall j\in R_k, i\neq j} c(P_k, j) \right) \right] + \sum_{k\in(X_k^i=0)} \left[r_i^k o_k c(i, NN_i^k) + w_i^k o_k c(i, P_k) \right], \tag{5}$$

which implies that if an agent replicates O_k (denoted in (5) as $k \in X_i^k = 1$), then the cost incurred due to reads is $0 = r_i^k o_k c(i, NN_i^k)$ since $NN_i^k = i$. The cost incurred due to local writes (or updates) is equal to zero since the copy resides locally, but whenever O_k is updated anywhere in the network, agent i has to continuously update O_k's contents locally as well. Therefore, the aggregate cost of writes is equivalent to $w_i^k o_k \sum_{\forall(j\in Rk), i\neq j} c(P_k, j)$. On the other hand if an agent does not replicate O_k (denoted in (5) as $k \in X_i^k = 0$), then the cost incurred due to reads is equal to $r_i^k o_k c(i, NN_i^k)$, and the cost incurred due to writes is equal to $w_i^k o_k c(i, P_k)$ since it only has to send the update to the primary server which then broadcasts the update based on R_k to the agents who have replicated the object.

Remarks Equation (5) captures the dilemma faced by an agent i when considering replicating O_k. If i replicates O_k then it brings down the read cost to zero, but now it has to keep the contents of O_k up to date. If i does not replicate O_k, then it reduces the overhead of keeping the contents up to date, but now it has to redirect the read requests to the nearest neighborhood server which holds a copy of O_k.

Truthful (Axiom 3) From Lemma 1, we know that truth-telling is a dominate strategy. From Axiom 2 we know that $t^i = CoR_k^i$.

Utilitarian (Axiom 4) We proceed in two steps.

1. Let $y = \{x^i, o\}$ and $y' = \{x^i, o'\}$. In the context of the data replication problem, the valuation of an agent is give as $v^i(x, t^i) = \sum_{k\in xi} CoR_k^i$. This means that when an agent i is asked to express its preference over two objects o and o', it can do so by calculating the object impact factor, i.e., the agent expresses its preference as $\min\{\sum_{k\in y} CoR_k^i, \sum_{k\in y'} CoR_k^i\}$.

2. A utilitarian mechanism is one that has an objective function that is the sum of all agents' valuations, i.e., $g(x,t) = \sum_i v^i(x,t^i) = \sum_i \sum_{k \in xi} CoR_k^i$.

We can see that Axiom 4 (Step 2) represents the data replication problem in its exact form as described in (4). For a minute, let us ponder over both the representations. Equation (4) expresses the fact that in the data replication problem we have to find object allocations such that the object transfer cost (OTC) is minimized. Step 2 (in conjunction with Step 1) of Axiom 4 does exactly the same, i.e., find the object allocations x^i such that the total cost of replication CoR_k^i is minimized.

Motivation (Axiom 5) The motivational payment for each agent i is defined as $p^i(t) = \sum_{k \in xi(t)} \min_{i' \neq i} t^{i'} = \sum_{k \in xi(t)} \min_{i' \neq i} CoR_k^{i'}$, i.e., for each object allocated to it, the agent is given payment equal to the overall second best cost of replication of any object to any server (a very strong incentive). The payment procedure also answers one of the pending questions of Axiom 1 (utility function), i.e., $u^i = \sum_{k \in xi(t)} \min_{i' \neq i} t^{i'} + v^i(t^i, x)$.

Remarks This motivational payment is need by the agents to cover the cost of hosting the object onto their server. This payment also ensures that the agents do indeed report true data. We justify this payment by analyzing the following cases:

1. *Over projection*: Agents in anticipation of more revenue over project their true data, but this does not help, as the agent who is allocated the object gets the second best payment.
2. *Under projection*: If every agent under projects their true data, that does not help either as the revenue would drop in proportion to the under projection.
3. *Random projection*: In this case the deserving agent would be at loss. Therefore, it is unlikely that a selfish agent would agree to project random true data.

For more details on the optimality of such type of payment procedure see [57]. In that paper, the authors have identified many such scenarios, but all fail to exploit this payment option.

Algorithmic output (Axiom 6) We now define an algorithm that actually aids the agents to execute their preferences. In the context of the data replication problem, the mechanism after gathering the true data from every agent, decides which object to be allocated to which agent. The mechanism is described in Fig. 2. This also answers the pending question of Axiom 1 (algorithmic output function). Before we describe the algorithm, the data replication problem in the axiomatic game theoretical mechanism form is given as follows:

> Find all the feasible outputs of the mechanism which are all the partitions $x = \{x^1 \cdots x^M\}$, where x is the entire replica allocation of the distributed system, and x^i is the set of replicas allocated to server i.
>
> 1. The objective function of the mechanism is $g(x,t) = \sum_i v^i(x,t^i)$.
> 2. Agent i's valuation is $v^i(x,t^i) = \sum_{k \in xi} t^i$.
> 3. Agent i's true data is $t^i = CoR_k^i$.

Axiomatic Game Theoretical Replica Allocation Mechanism (AGT-RAM)

Initialize:

LS, L^i, T^i_k, M, MT

WHILE $LS \neq$ NULL **DO**
  $OMAX =$ NULL; $MT =$ NULL; $P^i =$ NULL;
    PARFOR each $S^i \in LS$ **DO**
      FOR each $O_k \in L^i$ **DO**
        $T^i_k =$ compute(t^i_k); /*compute the valuation corresponding to the desired object*/
      ENDFOR
      $t^i_k = \text{argmax}_k(T^i_k)$;
      SEND t^i_k to M; **RECEIVE** at Mt^i_k in MT;
    ENDPARFOR
  $OMAX = \text{argmax}_k(MT)$; /*Choose the global dominate valuation*/
  DELETE k from MT;
  $P^i = \text{argmax}_k(MT)$; /*Calculate the payment*/
  BROADCAST $OMAX$;
  SEND P^i to S^i; /*Send payments to the agent who is allocate the object $OMAX$*/
  Replicate O_{OMAX};
  $b^i = b^i - o_k$; /*Update capacity*/
  $L^i = L^i - O_k$; /*Update the list*/
  IF $L^i =$ NULL **THEN SEND** info to M to update $LS = LS - S^i$; /*Update mechanism players*/
    PARFOR each $S^i \in LS$ **DO**
      Update NN^i_{OMAX} /*Update the nearest neighbor list*/
    ENDPARFOR /*Get ready for the next round*/
ENDWHILE

Fig. 2 Pseudo-code for AGT-RAM

4. Agent i's payment is $p^i(t) = \sum_{k \in xi(t)} \min_{i' \neq i} t^{i'}$.
5. Agent i's utility function is $u^i = p^i + v^i(t^i, x)$.

Remarks In the above problem formulation we did not mention that: (1) the agent i's valuation is actually to obtain the object impact (minimum cost of replication), (2) the agent i's server capacity constraint, and (3) the primary object constraints (both 2 and 3 are captured by x^i), but it is to be understood that they are indirectly embedded into the problem formulation.

Description of algorithm We maintain a list L^i at each server. This list contains all the objects that can be replicated by agent i onto server 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 server 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 servers 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^i_k, 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 following two results about the mechanism.

Theorem 4 *In the worst case AGT-RAM takes* $O(MN^2)$ *time.*

Proof The worst case scenario is when each server has sufficient capacity to store all objects. In that case, the while loop (Line 01) performs MN iterations. The time complexity for each iteration is governed by the two **PARFOR** loops (Lines 03 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 $O(MN^2)$. □

Theorem 5 *AGT-RAM is a truthful mechanism.*

Proof The algorithmic output is an allocation that minimizes the utilitarian function $\sum_i v^i(x, t^i)$. Let h^{-i} be $\sum_k \min_{i' \neq i} t^{i'}$, then $\sum_{i' \neq i} v^{i'}(x, t^{i'}) + h^{-i}$ is exactly the mechanism's payment function. It is also evident that truth-telling is the only dominate strategy. For simplicity let us consider the case for only one object. The argument for $k > 1$ is similar. Let d denote the declarations and t their real types. Consider the case where $d^i \neq t^i$ $(i = 1, 2)$. If $d^i > t^i$, then for d^{3-i} such that $d^i > d^{3-i} > t^i$, the utility for agent i is $t^i - d^i < 0$, which should have been zero in the case of truth-telling, but it is not. □

Theorem 6 *In the worst case AGT-RAM uses* $O(M^3N)$ *messages.*

Proof First we calculate the number of messages in a single iteration. First, each agent sends its true data to the mechanism, which constitutes M messages. Second, the mechanism broadcasts information about the object being allocated, this constitutes M messages. Third, the mechanism sends a single message about payment to the agent to whom the replica was assigned, which we can ignore since it has little impact on the total number of messages. The total number of messages required in a single iteration as of the order of M^2. From Theorem 4, we can conclude that in the worst case the mechanism requires $O(M^3N)$ messages. □

It is evident from Theorem 6 that the mechanism requires tremendous amounts of communications. This might be reduced by using some sophisticated network protocols. In the future generation distributed computing systems, the participating agents might actually be mobile, i.e., they can travel from node to node in the network and can converge to a specific server and execute the mechanism.

Next we present results that strengthen our claim on the optimality of the derived semi-distributed axiomatic mechanism. We begin by making the following observations. Assume that the mechanism $m = (x(b), p(b))$ is truthful and each payment $p^i(b^{-i}, b^i)$ and allocation $x^i(b^{-i}, b^i)$ is twice differentiable with respect to b^i, for all the values of b^{-i}. We fix some agent i and derive a formula for p^i, allocation x^i, and profit to be the functions of just agent i's bid b^i. Since agent i's profit is always maximized by bidding truthfully, the derivative is zero and the second derivative is non-positive at t^i. Since this holds no matter what the value of t^i is, we can integrate to obtain an expression for p^i. We state: $p^i(b^i) = p^i(0) + b^i x^i(b^i) - \int_0^{b^i} x^i(u)du$. This is now the basis of our extended theoretical results. Literature survey revealed the following two important characteristics of an optimal payment mechanism. We state them below.

Definition 6 With the other agents' bid b^{-i} fixed, consider $x^i(b^{-i}, b^i)$ as a single variable function of b^i. We call this the allocation curve or the allocation profile of agent i. We say the output function x is decreasing if each of the associated allocation curves is decreasing, i.e., $x^i(b^{-i}, b^i)$ is a decreasing function of b^i, for all i and b^{-i}.

Based on the above definition, we can state the following theorem.

Theorem 7 *A mechanism is truthful if its output function $x(b)$ is decreasing.*

Proof For simplicity we fix all bids b^{-i}, and focus on $x^i(b^{-i}, b^i)$ as a single variable function of b^i, i.e., the allocation x^i would only change if b^i is altered. We now consider two bids b^i and $b^{i'}$ such that $b^{i'} > b^i$. In terms of the true data t^i, this conforms to $CoR_k^{i'} > CoR_k^i$. Let x^i and $x^{i'}$ be the allocations made to the agent i when it bids b^i and $b^{i'}$, respectively. For a given allocation, the total replication cost associated can be represented as $C^i = \sum_{k \in x^i} CoR_k^i$. The proof of the theorem reduces to proving that $x^{i'} < x^i$, i.e., the allocation computed by the algorithmic output is decreasing in b^i. The proof is simple by contradiction. Assume that $x^{i'} \geq x^i$. This implies that $1/(C^i - CoR_k^i) < 1/(C^{i'} - CoR_k^i) \leq 1/(C^i - CoR_k^{i'})$. This means that there must be an agent $-i$ who has a bid that supersedes $b^{i'}$. But that is not possible as we began with the assumption that all other bids are fixed so there can be no other agent $-i$. If $i = -i$, then that is also not possible since we assumed that $b^{i'} > b^i$. □

We now extend the result obtained in Theorem 7 and state the following.

Theorem 8 *A decreasing output function admits a truthful payment scheme satisfying voluntary participation if and only if $\int_0^\infty x^i(b^{-i}, u)du < \infty$ for all i, b^{-i}. In this case we can take the payments to be: $p^i(b^{-i}, b^i) = b^i x^i(b^{-i}, b^i) + \int_0^\infty x^i(b^{-i}, u)du$.*

Proof The first term $b^i x^i(b^{-i}, b^i)$ compensates the cost incurred by agent i to host the allocation x^i. The second term $\int_0^\infty x^i(b^{-i}, u)du$ represents the expected profit of agent i. If agent i bids its true value t^i, then its profit is

$$= u^i(t^i, (b^{-i}, t^i))$$

$$= t^i x^i(b^{-i}, t^i) + \int_{t^i}^{\infty} x^i(b^{-i}, x)dx - t^i x^i(b^{-i}, t^i)$$
$$= \int_{t^i}^{\infty} x^i(b^{-i}, x)dx.$$

If agent i bids its true value, then the expected profit is greater than in the case it bids other values. We explain this as follows: If agent i bids higher ($b^{i'} > t^i$), then the expected profit is

$$= u^i(t^i, (b^{-i}, b^{i'}))$$
$$= b^{i'} x^i(b^{-i}, b^{i'}) + \int_{b^{i'}}^{\infty} x^i(b^{-i}, x)dx - t^i x^i(b^{-i}, b^{i'})$$
$$= (b^{i'} - t^i)x^i(b^{-i}, b^{i'}) + \int_{b^{i'}}^{\infty} x^i(b^{-i}, x)dx.$$

Because $\int_{b^{i'}}^{\infty} x^i(b^{-i}, x)dx < \infty$ and $b^{i'} > t^i$, we can express the profit when agent i bids the true value as follows: $\int_{t^i}^{b^{i'}} x^i(b^{-i}, x)dx + \int_{b^{i'}}^{\infty} x^i(b^{-i}, x)dx$. This is because x^i is decreasing in b^i and $b^{i'} > t^i$, we have the following equation: $(b^{i'} - t^i)x^i(b^{-i}, b^{i'}) < \int_{t^i}^{b^{i'}} x^i(b^{-i}, x)dx$. From this relation, it can be seen that the profit with overbidding is lower then the profit with bidding the true data. Similar arguments can be used for underbidding. □

6 Experiments and the discussion of results

We performed experiments on a 440 MHz Ultra 10 machine with 512 MB memory. The experimental evaluations were targeted to benchmark the placement policies. AGT-RAM was implemented using Ada and Ada GNAT's distributed systems annex GLADE [53].

To establish diversity in our experimental setups, the network connectively was changed considerably. We used four types of network topologies, which we explain below.

Pure random [61] (5 Topologies) A random graph $G(M, P(\text{edge} = p))$ with $0 \leq p \leq 1$ contains all graphs with nodes (servers) M in which the edges are chosen independently and with a probability p. Although this approach is extremely simple, yet it fails to capture significant properties of Web-like topologies [19]. The pure random topologies were obtained using GT-ITM [61] topology generator 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 1 kB of data and the latency on a link was assumed to be 2.8×10^8 m/s (copper wire). This reverse mapping technique is explained in a detailed fashion in [28].

Waxman model [59] (12 Topologies) The shortcomings of pure random topologies are overcome by using the Waxman model. In this method edges are added between pairs of nodes (u, v) with probability $P(u, v)$ that depends on the distance $d(u, v)$ between u and v. The Waxman model is given by:

$$P(u, v) = \beta e^{\frac{-d(u,v)}{L\alpha}},$$

where L is the maximum distance between any two nodes and $\alpha, \beta \in (0, 1]$. β is used to control the density of the graph. The larger the value of β the denser is the graph. α is used to control the connectively of the graph. The smaller the value of α the larger is the number of short edges [19]. The Waxman topologies were obtained using the values of $\alpha = \{0.1, 0.15, 0.2, 0.25\}$ and $\beta = \{0.2, 0.3, 0.4\}$.

Power-law model [48] (20 Topologies) The power-law model takes its inspiration from the Zipf law [62], and incorporates rank, out-degree and eigen exponents. We used Inet [10] topology generator that fully supports power-law based Web topologies. Briefly, Inet generates Autonomous System (AS) level topologies. These networks have similar if not the exact characteristics of the Internet from November 1997 to June 2000. The system takes in two parameters to generate topologies, namely: (1) the total number of nodes, and (2) the fraction (k) of degree-one nodes. Briefly, Inet starts form the total number of desired nodes and computes the number of months t it would take to grown the Internet from its size in November 1997 (which was 3037 nodes) to the desired number of nodes. Using t it calculates the growth frequency and the out-degree of the nodes in the network. This information is used to iteratively connect nodes till the required out-degree of nodes is reached. The power-law topologies were obtained using $k = \{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$.

Transit stub model [61] (12 Topologies) The Internet model at the autonomous system (AS) level can also be captured by using a hierarchical model. Authors in [61] derived a graph generation method using a hierarchical model in order to provide a more adequate router model of the Internet than the Waxman model. In their paper, each AS domain in the Internet was classified as either a *transit* domain or a *stub* domain, hence the name transit-stub model. In a stub domain, traffic between any two nodes u and v goes through that domain if and only if either u or v is in that domain. Contrarily, this restriction is relaxed in a transit domain. The GT-ITM topology generator [61] models a three-level hierarchy corresponding to transit domains, stub domain, and LANs attached to stub domains [19]. Using the GT-ITM topology generator, we generated 12 random transit-stub graphs with a total of 3718 nodes each, and then placed the primary server inside a randomly selected stub domain. In order to make the topologies as realistic as possible, we introduced routing delays to mimic routers' decision delays inside the core network. We set this delay to be equal to 20 ms/hop. In order to have a realistic upper bound on the self-injected delays, the maximum hop count between any pair of servers was limited to 14 hops.

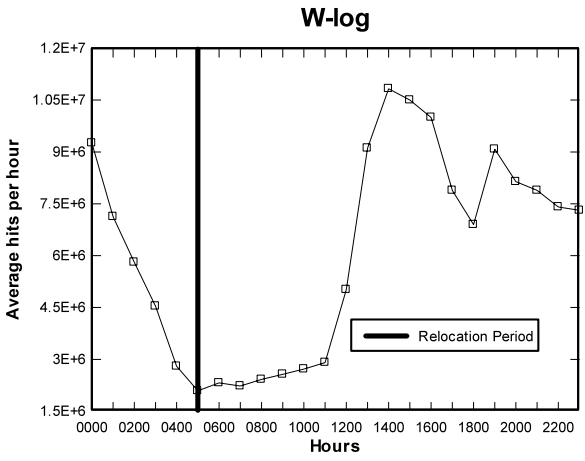
To evaluate the replica placement methods under realistic traffic patterns, we used the access logs collected at the Soccer World Cup 1998 website (W-log) [2] and

NASA Kennedy Space Center website (N-log) [50]. These two access logs compliment each other in many ways. The Soccer World Cup access log has over 1.35 billion requests, making it extremely useful to benchmark a given approach over a prolonged high access rate. The only drawback with these logs is that the users' IP addresses (that can potentially give us their approximate geographical locations) are replaced with an identifier. Although, we can obtain the information as to who were the top, say 500 users of the website, yet we cannot determine where the clients were from. To negate this drawback, we used the access logs collected at the NASA Kennedy Space Center website. These logs do not hide the IP addresses and thus the spatially skewed workload is preserved. Another benefit of the Space Center's log is that the access requests are very concentrated, i.e., a majority of the access request are sent from few clients (or cluster of clients)—capturing the temporal diversity of the users.

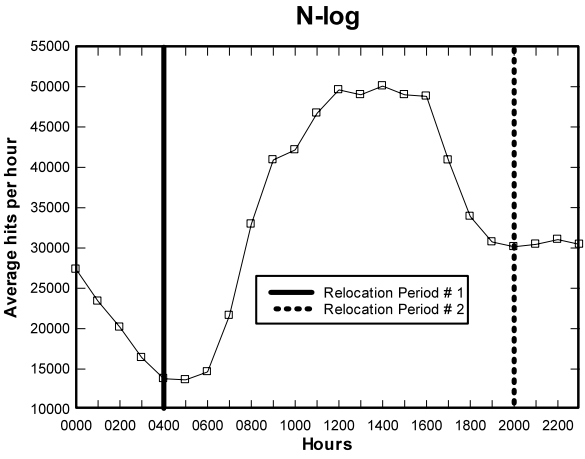
We used 88 days of the Soccer World Cup 1998 access logs, i.e., the (24 hours) logs from April 30, 1998 to July 26, 1998. We used 31 days of the NASA Kennedy Space Center access logs, i.e., the (24 hours) logs from July 1, 1995 to July 31, 1995. To process the logs, we wrote a script that returned: only those objects that 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 $1.5 \times$ *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.

Because the access logs are of the year 1998, we first use Inet 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 $M = 3718$. Due to space limitations, we do not show the detailed results obtained using every topology. However, we do provide the averaged performance of all the comparative algorithms over all the 49 topologies and 119 24 hours access log. Thus, each data point in the plots represents the average performance of 5831 ($= 49 \times 119$) data points.

Determination of the relocation period As noted previously (in Sect. 3), the time (interval t) when to initiate the replica placement techniques requires high-level human intervention. Here, we will show that this parameter if not totally can at least partially be automated. The decision when to initiate the replica placement techniques depend on the past trends of the user access patterns. Figures 3(a) and 3(b) show the average (over the entire access log) user access patterns extracted from the W-log



(a)



(b)

Fig. 3 (**a**) User access pattern extracted from W-log. (**b**) User access pattern extracted from N-log

and N-log. From Fig. 3(a) we can clearly see that the Soccer World Cup 1998 website incurred soaring and stumpy traffic at various intervals during the 24-hour time period (it is to be noted that the W-log has a time stamp of GMT + 1). For example, the website records its minimum requests at 0500 hours. This would be an ideal time to invoke the replica placement technique(s), since the traffic is at its minimum and fewer users will be affected by the relocation of data objects in the network. Another potential time period for invoking the replica placement technique(s) is at 1800 hours. In our experiments we did not use this time period since the volume of traffic at 1800 hours is enormous and it immediately soars; thus, leaving little buffer time for the completion of the replica placement technique(s).

On the other hand, the analysis of N-log (it is to be noted that the N-log has a time stamp of GMT-4) reveals two periods where the traffic drops to minimum, i.e., at 0400 hours and at 2000 hours. This is denoted by the two vertical lines in Fig. 3(b).

Therefore, for the N-log a replica placement algorithm could be initiated twice daily: (1) at 0400 hours and (2) at 2000 hours. The time interval t for 0400 hours would be $t = (2000–0400) = 6$ hours and for 2000 hours $t = (0400–2200) = 18$ hours. For the W-log a replica placement algorithm could be initiated once daily at 0500 hours. The time interval t for 0500 hours would be $t = (0500–0500) = 24$ hours.

6.1 Comparative algorithms

For comparisons, we selected ten various types of replica placement techniques. To provide a fair comparison, the assumptions and system parameters were kept the same in all the approaches. For fine-grained replication, the algorithms proposed in [28, 29, 42, 45, 54] are the only ones that address the problem domain similar to ours. We select from [54] the greedy approach (Greedy) for comparison because it is shown to be the best compared with four other approaches (including the proposed technique in [42]); thus, we indirectly compare with four additional approaches as well. Algorithms reported in [29] (the efficient branch and bound based technique Aε-Star), [28] (Dutch (DA) and English auctions (EA)) and [45] (Genetic based algorithm (GRA)) are also among the chosen techniques for comparisons. More recently, researchers have proposed unique and sophisticated solutions to the traditional data replication problem as reported by the authors of this paper in [45] by taking into account various parameterization. Among these newly proposed solutions we selected for comparison with the proposed AGT-RAM procedure: (1) the data-aware storage allocation algorithm (DASA) [8], (2) the tree inspired constructs FindR (FR) and Min-Max-Load (MML) [43], (3) the procedure based on Benders' decomposition (BD) [4], and (4) the game theoretical procedure Distributed Selfish Replication (DSR) [41]. We encourage the readers to obtain an insight on the comparative techniques from the referenced papers.

Performance metric The solution quality is measured in terms of network communication cost (OTC percentage) that is saved under the replication scheme found by the algorithms, compared to the initial one, i.e., when only primary copies exists.

6.2 Comparative analysis

We study the behavior of the placement techniques when the number of servers increases (Fig. 4), by setting the number of objects to 25,000, while in Fig. 6, we study the behavior when the number of objects increase, by setting the number of servers to 3718. For the first experiment we fixed $C = 30\%$ and $R/W = 0.25$. We intentionally chose a high workload so as to see if the techniques studied successfully handled the extreme cases. By adding a server in the network, we introduce additional traffic due to its local requests, together with more storage capacity to be used for replication. AGT-RAM balances and explores these diverse effects, so as to achieve highest OTC savings. GRA and MML showed the worst performance along all of the techniques. We must note that all of the heuristics showed an initial gain because with the increase in the number of servers the data object placement permutations increase exponentially. However, with the further increase in the number of servers this phenomenon

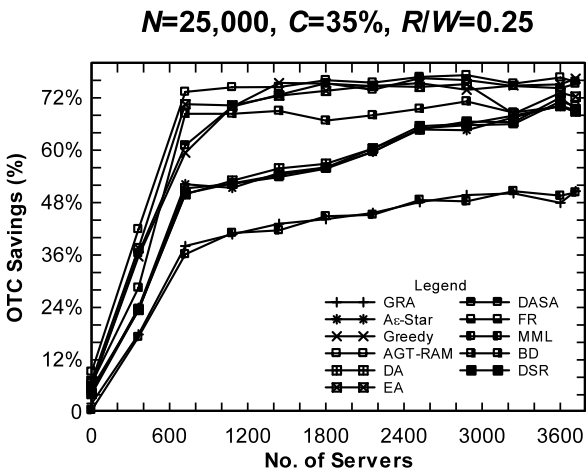


Fig. 4 OTC savings versus number of servers

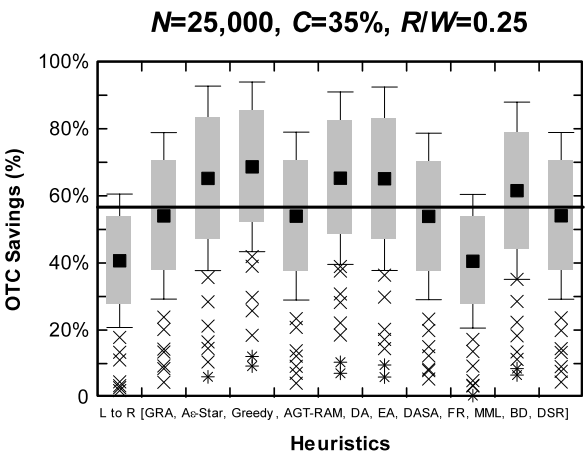


Fig. 5 Average performance of the heuristics (OTC savings versus number of servers). $\top$ (Mean + 1.5 ∗ Std Dev), $\perp$ (Mean − 1.5 ∗ Std Dev), ▒ (Mean + Std Dev–Mean − Std Dev), ■ (Mean), × (Outliers), ✳ (Extremes), ▬ (Grand mean)

is unobservable because all of the essential data objects are already replicated. The top performing techniques (AGT-RAM, Greedy, and BD) showed an almost constant performance increase (after the initial surge in OTC savings). GRA also showed a similar trend but maintained lower OTC savings. This was in line with the claims presented in [28] and [45]. To further clarify the relative performance of the comparative techniques, Fig. 5 plots the mean performance of the heuristics with bars at the maximum and minimum limits with values of mean +1.5 times the standard deviation and mean −1.5 times the standard deviation, respectively. The shaded block represents the maximum and minimum limits with values of mean + standard deviation and mean − standard deviation, respectively. The solid line across the plots is the grand mean, the solid block (■) represents the mean, the cross (×) represents the outliers, and the asterisk (✳) denotes the extremes. We limit the outliers and extremes to 2 and 3 standard deviations, respectively. The plots are self-explanatory and show exactly which algorithms provide high (consistent) performance. The performance of

the techniques based on the OTC versus number of servers criteria are ranked as follows: (1) AGT-RAM; (2) Greedy; (3) EA; (4) DASA; (5) BD; (6) Aε-Star; (7) DSR; (8) DA; (9) FR; (10) GRA; (11) MML.

To observe the effect of increase in the number of objects in the system, we chose a softer workload with $C = 20\%$ and $R/W = 0.70$. The intention was to observe the trends for all the algorithms under various workloads. The increase in the number of objects has diverse effects on the system as new read/write patterns (since the users are offered more choices) emerge, and also the strain on the overall storage capacity of the system increases (due to the increase in the number of replicas). An effective replica allocation method should incorporate both the opposing trends. From the plot, the most surprising result came from BD. The heuristic dropped its savings from 73% to 13%. Moreover, the GRA heuristic also dropped its savings from 35% to 0.015. This was contradictory to what was reported in [45]. However, we justify such a contradictory behavior by observing that the authors of the GRA heuristic had used a uniformly distributed link cost topology, and the traffic pattern was based on the Zipf distribution [62]. The traffic access logs of the Soccer World Cup 1998 and the NASA Kennedy Space Center are more or less double-Pareto in nature [1]. In either case the exploits and limitations of the technique under discussion are obvious. The plot also shows a near identical performance by AGT-RAM and BD for the smaller number of data objects. The relative difference among the two techniques was less than 2%. However, AGT-RAM does maintain its dominance and it is vividly clear towards the larger increase in the number of data objects. To better understand the relative performance of the comparative techniques, readers are encouraged to examine the trends observable from Fig. 7. The performance of the techniques based on the OTC versus the number of objects criteria are ranked as follows: (1) AGT-RAM; (2) BD; (3) Greedy; (4) DASA; (5) FR; (6) DSR; (7) DA; (8) EA; (9) Aε-Star; (10) GRA; (11) MML.

Next, 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 Fig. 8, which shows the performance of the algorithms. GRA once again performed the worst. The gap between all other approaches was reduced to within 20% of each other. AGT-RAM, EA, and FR 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 of 39% from 1.7% to 41% followed by DRA, and DASA. Further experiments with various update ratios showed similar plot trends. The performance of the comparative techniques based on the OTC versus system capacity criteria can be observed from Fig. 9. The relative performance of the heuristics can be ranked: (1) AGT-RAM; (2) Greedy; (3) FR; (4) EA; (5) DA; (6) MML; (7) BD; (8) DSR; (9) DASA; (10) Aε-Star; (11) GRA.

Next, we observe the effects of increase in the read and write frequencies. Because these two parameters are complementary to each other, we describe them together.

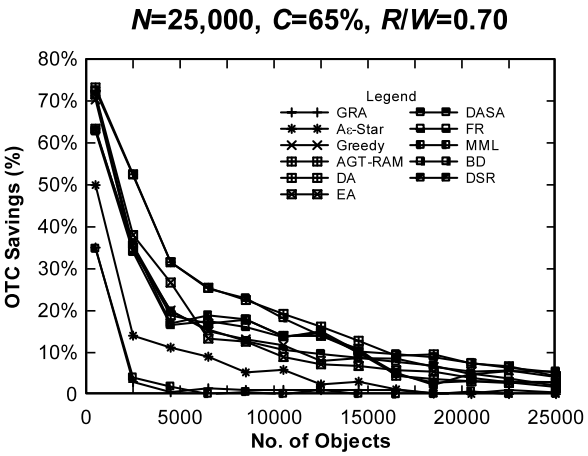


Fig. 6 OTC savings versus number of objects

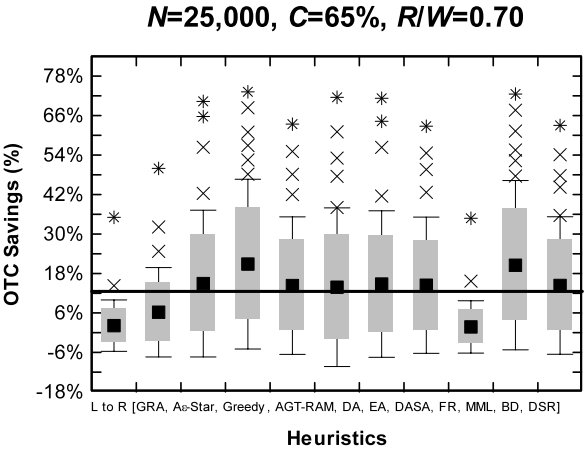


Fig. 7 Average performance of the heuristics (OTC savings versus number of objects). ⊤ (Mean + 1.5 ∗ Std Dev), ⊥ (Mean − 1.5 ∗ Std Dev), ▒ (Mean + Std Dev–Mean − Std Dev), ■ (Mean), × (Outliers), ✳ (Extremes), ▬ (Grand mean)

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 Fig. 10 shows the results of read/write ratio against the OTC savings. A clear classification can be made between the algorithms. AGT-RAM and EA incorporate the increase in the number of reads by replicating more objects and thus savings increased up to 89%, while BD gained the least of the OTC savings of up to 46%. Figure 11 depicts the relative average performance of all of the comparative heuristics. The performance

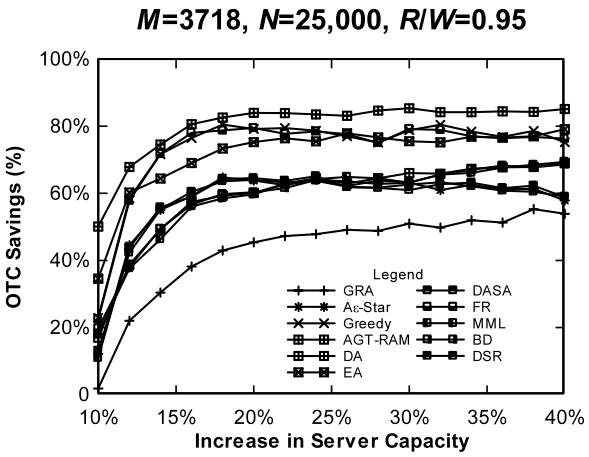


Fig. 8 OTC savings versus capacity

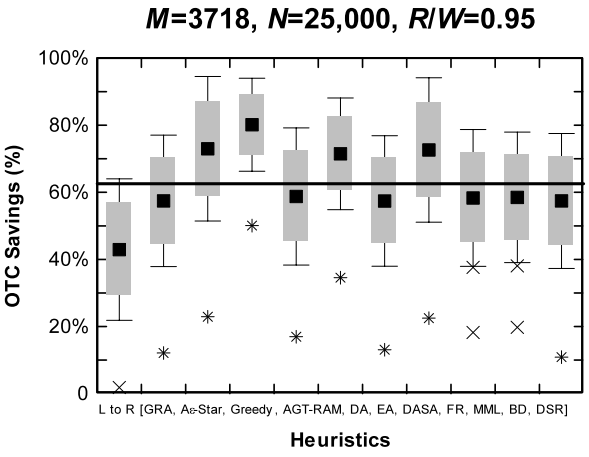


Fig. 9 Average performance of the heuristics (OTC savings versus capacity). ⊤ (Mean + 1.5 ∗ Std Dev), ⊥ (Mean − 1.5 ∗ Std Dev), ▒ (Mean + Std Dev–Mean − Std Dev), ■ (Mean), × (Outliers), ✳ (Extremes), ━ (Grand mean)

of the heuristics based on the OTC versus R/W ratio criteria are ranked as follows: (1) AGT-RAM; (2) Greedy; (3) DASA; (4) MML; (5) DSR; (6) Aε-Star; (7) FR; (8) DA; (9) EA; (10) BD; (11) GRA.

Lastly, we compare the termination time of the algorithms. Various problem instances were recorded with randomly chosen C and R/W values. For all instances, the number of servers and data objects was kept constant to the maximum load of 3718 and 25,000, respectively. The entries in Table 3 made bold represent the fastest time recorded over the problem instance. It is observable that AGT-RAM terminated faster than all the other techniques, followed by Greedy, DA, EA, Aε-Star, and GRA.

In summary, the replica allocation methods can be ranked (based on the previous four rankings) as follows: (1) AGT-RAM; (2) Greedy; (3) DASA; (4) EA; FR; BD (7) DSR; (8) DA; (9) Aε-Star; (10) MML; (11) GRA.

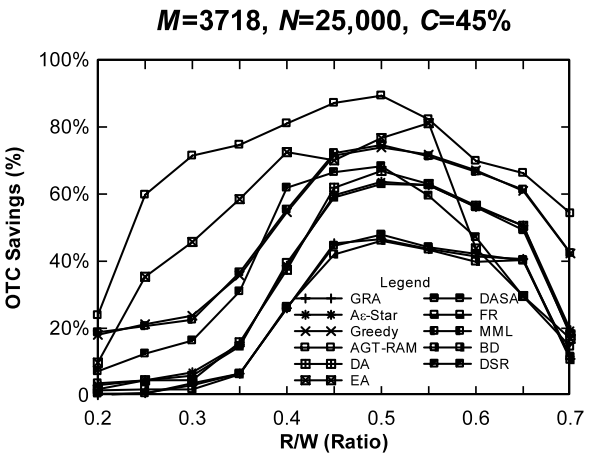


Fig. 10 OTC savings versus read/write ratio

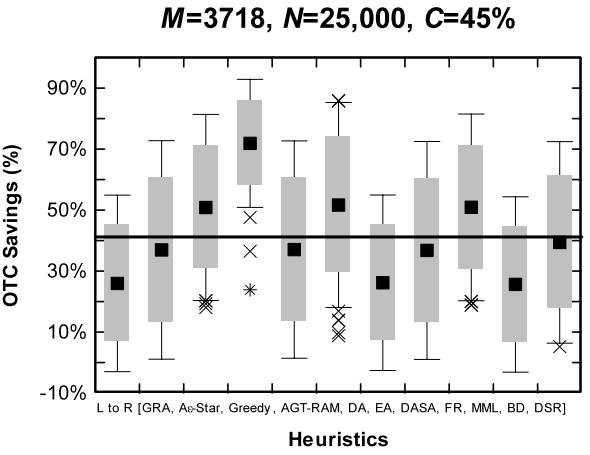


Fig. 11 Average performance of the heuristics (OTC savings versus read/write ratio). ⊤ (Mean + 1.5 ∗ Std Dev), ⊥ (Mean − 1.5 ∗ Std Dev), ▒ (Mean + Std Dev–Mean − Std Dev), ■ (Mean), × (Outliers), ✳ (Extremes), ━ (Grand mean)

Table 3 Running time of the replica placement methods in seconds

GRA	Aε-Star	Greedy	AGT-RAM	DA	EA	DASA	FR	MML	BD	DSR
496.00	406.70	314.53	**194.78**	345.19	364.78	498.28	495.89	494.32	500.19	457.67
566.02	450.14	336.44	**206.43**	355.21	371.62	569.95	563.69	573.50	598.08	571.00
569.92	467.46	361.24	**242.54**	369.09	397.94	577.26	586.12	571.97	554.90	595.00
677.47	500.28	456.49	**290.95**	479.32	493.08	678.38	674.40	676.63	687.45	678.90
732.20	509.15	475.54	**288.10**	498.84	540.19	763.01	731.28	733.31	734.04	735.39
794.14	546.55	479.13	**314.40**	503.68	545.34	793.61	800.74	796.62	701.68	796.90
890.58	755.57	619.11	**371.43**	670.92	679.20	875.96	885.67	885.10	852.66	888.72
908.82	779.04	633.62	**407.74**	704.20	702.84	919.96	900.94	906.61	956.89	904.86
934.69	883.67	653.98	**408.62**	722.52	753.15	932.45	936.91	937.10	939.15	936.10

7 Concluding remarks

This paper proposed a semi-distributed axiomatic game theoretical replica allocation mechanism (AGT-RAM) for distributed database systems. AGT-RAM is a protocol for automatic replication and migration 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 procedure of AGT-RAM was designed such that each server was required to present a list of data objects that if replicated onto that server would bring the communication cost to its minimum. These lists were reviewed at the central decision body that gave the final decision as to what object are to be replicated onto what servers. This semi-distributed procedure takes away all the heavy processing from the central decision making body and gives it to the individual servers. For each object, the central body is only required to make a binary decision: (0) not to replicate or (1) to replicate.

To compliment our theoretical results, we compared AGT-RAM with ten replica allocation methods, namely, branch and bound, greedy, genetic, data-aware replication, tree inspired bottom-up procedure, tree inspired min-max procedure, Benders' decomposition based procedure, game theoretical English auction, game theoretical Dutch auction, and game theoretical selfish replication procedure. The experimental setups were designed in such a fashion that they resembled real world scenarios. We employed GT-ITM and Inet to gather 42 various Internet topologies and used the traffic logs collected at the Soccer World Cup 1998 website and NASA Kennedy Space Center for mimicking user access requests. The experimental study revealed that the proposed AGT-RAM 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 turnaround time. Fourth, the complexity of the data replication problem was decreased by multifold. AGT-RAM limits the complexity by partitioning the complex global problem of replica allocation, into a set of simple independent sub problems. This approach is well suited to the large distributed computing systems that are composed of autonomous agents which do not necessarily cooperate to improve the system wide goals. All the above improvements were achieved by a simple semi-distributed AGT-RAM.

As future work, we would like to extend the semi-distributed model to regional autonomous, self-governed and self-repairing 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 objects. Moreover,

severs may become overloaded whence numerous clients are requesting services to distributed data objects. We also intent to investigate load balancing in distributed replicated databases.

Acknowledgement A preliminary version of this paper appeared as S.U. Khan and I. Ahmad, "A Semi-Distributed Axiomatic Game Theoretical Mechanism for Replicating Data Objects in Large Distributed Computing Systems," in *21st International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, CA, March 2007.

References

1. Ahmad, I., Ghafoor, A.: Semi-distributed load balancing for massively parallel multicomputer systems. IEEE Trans. Softw. Eng. **17**(10), 987–1004 (1991)
2. Arlitt, M., Jin, T.: Workload characterization of the 1998 World Cup Web Site. Tech. Report, Hewlett Packard Lab, Palo Alto, HPL-1999-35(R.1) (1999)
3. Awerbuch, B., Bartal, Y., Fiat, A.: Competitive distributed file allocation. In: Proc. 25th ACM Symposium on Theory of Computation, pp. 164–173 (1993)
4. Bekta, T., Cordeaua, J.-F., Erkutc, E., Laportea, G.: Exact algorithms for the joint object placement and request routing problem in content distribution networks. Comput. Oper. Res. **35**, 3860–3884 (2008)
5. Briest, P., Krysta, P., Vöcking, B.: Approximation techniques for utilitarian mechanism design. In: Proc. of 37th ACM Symposium on Theory of Computation, pp. 39–48 (2005)
6. Campbell, D.: Resource Allocation Mechanisms. Cambridge University Press, Cambridge (1987)
7. Casey, R.: Allocation of copies of a file in an information network. In: Proc. Spring Joint Computer Conf., IFIPS, pp. 617–625 (1972)
8. Chandy, J.A.: A generalized replica placement strategy to optimize latency in a wide area distributed storage system. In: International Workshop on Data-Aware Distributed Computing, pp. 49–54 (2008)
9. Chandy, K., Hewes, J.: File allocation in distributed systems. In: Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, pp. 10–13 (1976)
10. Chang, H., Govindan, R., Jamin, S., Shenker, S.: Towards capturing representative AS-level Internet topologies. Comput. Netw. J. **44**(6), 737–755 (2004)
11. Chu, W.: Optimal file allocation in a multiple computer system. IEEE Trans. Comput. **C-18**(10), 885–889 (1969)
12. Chun, B.-G., Chaudhuri, K., Wee, H., Barreno, M., Papadimitriou, C., Kubiatowicz, J.: Selfish caching in distributed systems: a game-theoretic analysis. In: Proc. of 23rd ACM Symposium on Principles of Distributed Computing, pp. 21–30 (2004)
13. Cidon, I., Kutten, S., Soffer, R.: Optimal allocation of electronic content. In: Proc. of IEEE INFOCOM, pp. 1773–1780 (2001)
14. Cook, S., Pachl, J., Pressman, I.: The optimal location of replicas in a network using a READ-ONE-WRITE-ALL policy. Distrib. Comput. **15**(1), 57–66 (2002)
15. Dowdy, L., Foster, D.: Comparative models of the file assignment problem. ACM Comput. Surv. **14**(2), 287–313 (1982)
16. Eswaran, K.P: Placement of records in a file and file allocation in a computer. In: Proc. of IFIP Congress, pp. 304–307 (1974)
17. Green, J., Laffont, J.: Characterization of satisfactory mechanisms for the revelation of preferences for public goods. Econometrica **45**(2), 427–438 (1977)
18. Groves, T.: Incentives in teams. Econometrica **41**, 617–631 (1973)
19. Habegger, P., Bieri, H.: Modeling the topology of the Internet: an assessment. Tech. Report, Institut für Informatik und angewandte Mathematik, Universität Bern, IM-02-2002
20. Hakimi, S.: Optimum location of switching centers and the absolute centers and medians of a graph. Oper. Res. **12**, 450–459 (1964)
21. Hara, T.: Effective replica allocation in ad hoc networks for improving data accessibility. In: Proc. of INFOCOM, pp. 1568–1576 (2001)
22. Heddaya, A., Mirdad, S.: WebWave: globally load balanced fully distributed caching of hot published documents. In: Proc. 17th International Conference on Distributed Computing Systems, pp. 160–168 (1997)

23. Jamin, S., Jin, C., Jin, Y., Riaz, D., Shavitt, Y., Zhang, L.: On the placement of Internet instrumentation. In: Proc. of the IEEE INFOCOM, pp. 295–304 (2000)
24. Jamin, S., Jin, C., Kurc, T., Raz, D., Shavitt, Y.: Constrained mirror placement on the Internet. In: Proc. of the IEEE INFOCOM, pp. 31–40 (2001)
25. Kalpakis, K., Dasgupta, K., Wolfson, O.: Optimal placement of replicas in trees with read, write, and storage costs. IEEE Trans. Parallel Distrib. Syst. **12**(6), 628–637 (2001)
26. Kangasharju, J., Roberts, J., Ross, K.: Object replication strategies in content distribution networks. In: Proc. of Web Caching and Content Distribution Workshop, pp. 455–456 (2001)
27. Karlsson, M., Mahalingam, M.: Do we need replica placement algorithms in content delivery networks? In: Proc. of Web Caching and Content Distribution Workshop, pp. 117–128 (2002)
28. Khan, S., Ahmad, I.: Internet content replication: a solution from game theory. Technical Report, University of Texas at Arlington, CSE-2004-5 (2004)
29. Khan, S., Ahmad, I.: Heuristic-based replication schemas for fast information retrieval over the Internet. In: Proc. of 17th International Conference on Parallel and Distributed Computing Systems, pp. 278–283 (2004)
30. Khan, S., Ahmad, I.: A powerful direct mechanism for optimal WWW content replication. In: Proc. of 19th IEEE International Parallel and Distributed Processing Symposium (2005)
31. Khan, S.U., Ahmad, I.: RAMM: a game theoretical replica allocation and management mechanism. In: 8th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN), Las Vegas, NV, USA, pp. 160–165, December 2005
32. Khan, S.U., Ahmad, I.: A pure nash equilibrium guaranteeing game theoretical replica allocation method for reducing web access time. In: 12th International Conference on Parallel and Distributed Systems (ICPADS), Minneapolis, MN, USA, pp. 169–176, July 2006
33. Khan, S.U., Ahmad, I.: Replicating data objects in large-scale distributed computing systems using extended Vickery auction. Int. J. Comput. Intell. **3**(1), 14–22 (2006)
34. Khan, S.U., Ahmad, I.: Discriminatory algorithmic mechanism design based WWW content replication. Informatica **31**(1), 105–119 (2007)
35. Khan, S.U., Ahmad, I.: A cooperative game theoretical replica placement technique. In: 13th International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, Taiwan, December 2007
36. Korupolu, M., Plaxton, C.: Analysis of a local search heuristic for facility location problems. J. Algorithms **37**(1), 146–188 (2000)
37. Krick, C., Racke, H., Westermann, M.: Approximation algorithms for data management in networks. In: Proc. of the Symposium on Parallel Algorithms and Architecture, pp. 237–246 (2001)
38. Krishna, V.: Auction Theory. Academic Press, San Diego (2002)
39. Krishnan, P., Raz, D., Shavitt, Y.: The cache location problem. IEEE/ACM Trans. Netw. **8**(5), 568–582 (2000)
40. Kurose, J., Simha, R.: A microeconomic approach to optimal resource allocation in distributed computer systems. IEEE Trans. Comput. **38**(5), 705–717 (1989)
41. Laoutaris, N., Telelis, O., Zissimopoulos, V.: Distributed selfish replication. IEEE Trans. Parallel Distrib. Syst. **17**(12), 1401–1413 (2006)
42. Li, B., Golin, M., Italiano, G., Deng, X.: On the optimal placement of web proxies in the Internet. In: Proc. of the IEEE INFOCOM, pp. 1282–1290 (2000)
43. Lin, Y.-F., Liu, P., Wu, J.-J.: Optimal placement of replicas in data grid environments with locality assurance. In: 12th International Conference on Parallel and Distributed Systems (ICPADS), pp. 465–474 (2006)
44. Loukopoulos, T., Papadias, D., Ahmad, I.: An overview of data replication on the Internet. In: Proc. of International Symposium on Parallel Architectures, Algorithms and Networks, pp. 31–38 (2002)
45. Loukopoulos, T., Ahmad, I.: Static and adaptive distributed data replication using genetic algorithms. J. Parallel Distrib. Comput. **64**(11), 1270–1285 (2004)
46. Mahmoud, S., Riordon, J.: Optimal allocation of resources in distributed information networks. ACM Trans. Database Syst. **1**(1), 66–78 (1976)
47. Mas-Collel, A., Whinston, W., Green, J.: Microeconomic Theory. Oxford University Press, London (1995)
48. Medina, A., Matta, I., Byers, J.: On the origin of power laws in Internet topologies. ACM Comput. Commun. Rev. **30**(2), 18–28 (2000)
49. Narebdran, B., Rangarajan, S., Yajnik, S.: Data distribution algorithms for load balancing fault-tolerant web access. In: Proc. of the 16th Symposium on Reliable Distributed Systems, pp. 97–106 (1997)

50. NASA Kennedy Space Center access log. Available at: http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html
51. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proc. of 31st ACM Symposium on Theory of Computation, pp. 129–140 (1999)
52. Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
53. Pautet, L., Tardieu, S.: GLADE: a framework for building large object-oriented real-time distributed systems. In: 3rd International Symposium on Object-Oriented Real-Time Distributed Systems, pp. 244–251 (2000)
54. Qiu, L., Padmanabhan, V., Voelker, G.: On the placement of web server replicas. In: Proc. of the IEEE INFOCOM, pp. 1587–1596 (2001)
55. Rabinovich, M.: Issues in Web content replication. Data Eng. Bull. **21**(4), 21–29 (1998)
56. Radoslavov, P., Govindan, R., Estrin, D.: Topology-informed Internet replica placement. Comput. Commun. **25**(4), 384–392 (2002)
57. Saurabh, S., Parkes, D.: Hard-to-manipulate VCG-based auctions. Available at: http://www.eecs.harvard.edu/econcs/pubs/hard_to_manipulate.pdf
58. Venkataramanj, A., Weidmann, P., Dahlin, M.: Bandwidth constrained placement in a WAN. In: Proc. ACM Symposium on Principles of Distributed Computing, pp. 134–143 (2001)
59. Waxman, B.: Routing of multipoint connections. IEEE J. Sel. Areas Commun. **6**(9), 1617–1622 (1988)
60. Wolfson, O., Jajodia, S., Hang, Y.: An adaptive data replication algorithm. ACM Trans. Database Syst. **22**(4), 255–314 (1997)
61. Zegura, E., Calvert, K., Donahoo, M.: A quantitative comparison of graph-based models for Internet topology. IEEE/ACM Trans. Netw. **5**(6), 770–783 (1997)
62. Zipf, G.: Human Behavior and the Principle of Least-Effort. Addison-Wesley, Reading (1949)