Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 64 (2004) 1270–1285
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Static and adaptive distributed data replication using genetic algorithms

Thanasis Loukopoulos[a], Ishfaq Ahmad[b,*]

[a] *Department of Computer Science, The Hong Kong University of Science and Technology, Kowloon, Hong Kong*
[b] *Department of Computer Science and Engineering, The University of Texas at Arlintgon, P.O. Box 19015, 248 D/E Nedderman Hall, 416 Yates St., Arlington, TX 16019-0015, USA*

Received 4 May 2001; received in revised form 12 March 2004

Abstract

Fast dissemination and access of information in large distributed systems, such as the Internet, has become a norm of our daily life. However, undesired long delays experienced by end-users, especially during the peak hours, continue to be a common problem. Replicating some of the objects at multiple sites is one possible solution in decreasing network traffic. The decision of what to replicate where, requires solving a constraint optimization problem which is NP-complete in general. Such problems are known to stretch the capacity of a Genetic Algorithm (GA) to its limits. Nevertheless, we propose a GA to solve the problem when the read/write demands remain static and experimentally prove the superior solution quality obtained compared to an intuitive greedy method. Unfortunately, the static GA approach involves high running time and may not be useful when read/write demands continuously change, as is the case with breaking news. To tackle such case we propose a hybrid GA that takes as input the current replica distribution and computes a new one using knowledge about the network attributes and the changes occurred. Keeping in view more pragmatic scenarios in today's distributed information environments, we evaluate these algorithms with respect to the storage capacity constraint of each site as well as variations in the popularity of objects, and also examine the trade-off between running time and solution quality.

Keywords: Data replication; Genetic algorithms; Static and dynamic allocation; Internet; World wide web; Greedy method

1. Introduction

Data replication across a read intensive network can potentially lead in increased traffic savings which can indirectly reduce the response time experienced by end-users. On the other hand in the presence of updates, maintaining large number of copies can be a performance bottleneck. There are three main issues when talking about replication in traditional distributed systems namely, allocation [15], consistency [2] and fault tolerance [3]. Recently, the high download times experienced during peak hours in the www [9], spurred interest on replicating data objects over the web [7,23,30] and providing mechanisms to allow user requests to be redirected towards a geographically close replica [28]. Under the web context, consistency and fault tolerance are not of primary importance while allocation still remains a challenge. Surprisingly, the current trend is to perform coarse grain replication in the form of server mirroring, i.e., duplicating all the contents of a web server [27]. As the web grows larger and larger and caching limitations become apparent [1,10] it is reasonable to expect that the importance of fine grain replication, e.g., replicating pages, will increase.

The first standpoint from where we address the problem of data replication is a "cold" network start where no replicas exist and the read and update frequencies remain static. The target is to define an appropriate replica distribution that minimizes the network traffic. We formulate the problem as a constraint optimization one and show that the relevant decision problem is NP-complete. We then propose a greedy heuristic called simple replication algorithm (SRA) and a genetic algorithm-based heuristic called genetic replication algorithm (GRA).

* Corresponding author. Fax: +1-817-272-3784.
E-mail address: iahmad@cse.uta.edu (I. Ahmad).

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2004.04.005

We evaluate both algorithms extensively and compare them against random search for the case where each object has equal probability to be accessed by any of the sites in the network, as well as when the request/site pattern follows a normal distribution. Linear Programming (LP) is used to estimate an upper bound for the optimal solution resulting by exhaustive search, while the integer parts of the solution found by LP (Linear Integer Programming LIP) are evaluated in order to define a lower bound for the optimal. Results show that in most cases both algorithms achieve performance in between these two bounds, with GRA constantly outperforming SRA, especially under the uniform assumption for the requests.

The second data replication aspect we tackle with, is the case where replica allocation was already performed but due to changes in the exhibited requests adaptations are needed. Both SRA and GRA are static algorithms. Once the replica distribution for the objects is obtained, changes in access patterns would result in the algorithms being run from scratch, which is potentially time consuming. For this reason we propose an extension of the GRA algorithm called Adaptive GRA (AGRA) that given an already realized replication scheme and the changes in read and write requests for particular objects, it quickly adapts the replica distribution to reflect the new demands. The reduced search space on which AGRA operates allows it to provide high solution quality within acceptable for a dynamic environment time limits.

The rest of the paper (a smaller version of which appeared in [24]) is organized as follows: Section 2 elaborates the problem and describes a cost model for the total data transfer cost. The same section includes the NP-completeness of the problem. Section 3 describes the greedy and evolutionary static methods, while Section 4 introduces AGRA. Sections 5 and 6, respectively, present the experimental results and the related work. Section 7 includes the concluding remarks.

2. The data replication problem (DRP)

First, we describe the inputs to (DRP) and introduce a notation (see Table 1) that will be subsequently used.

Consider a distributed system comprising M sites, with each site having its own processing power, memory and storage media. Let $S^{(i)}$, $s^{(i)}$ be the name and the total storage capacity (in simple data units e.g. blocks), respectively, of site i where $1 \leq i \leq M$. The M sites of the system are connected by a communication network. A link between two sites $S^{(i)}$ and $S^{(j)}$ (if it exists) has a positive integer $C(i,j)$ associated with it, giving the communication cost for transferring a data unit between sites $S^{(i)}$ and $S^{(j)}$. If the two sites are not directly connected by a communication link then the above cost is given by the sum of the costs of all the links in a chosen path from site $S^{(i)}$ to site $S^{(j)}$. We assume that $C(i, j) = C(j, i)$ and is known a priori to represent the cumulative cost of the shortest path between $S^{(i)}$ and $S^{(j)}$. Let there be N objects, named $\{O_1, O_2, \ldots, O_N\}$. The size of object O_k is denoted by o_k and is measured in simple data units. 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.

Table 1

Symbol	Meaning
O_k	kth object
o_k	Size of object k
$S^{(i)}$	ith site
$s^{(i)}$	Total storage capacity of $S^{(i)}$
$b^{(i)}$	Remaining storage capacity of $S^{(i)}$
M	Number of sites in the network
N	Number of objects in the distributed system
$r_k^{(i)}$	Number of reads from site i for object k
$R_k^{(i)}$	Associated cost of the $r_k^{(i)}$ reads
$w_k^{(i)}$	Number of writes from site i for object k
$W_k^{(i)}$	Associated cost of the $w_k^{(i)}$ writes
$C(i,j)$	Communication cost (per unit) between sites i and j
SP_k	Primary site of kth object
$SN_k^{(i)}$	Nearest site of site i, which holds object k
R_k	Replication scheme of the kth object
D	Total data transfer cost function
$B_k^{(i)}$	Benefit value, that is, the NTC saves we can achieve by replicating the jth object at the ith site.

The replication policy assumes the existence of one primary copy for each object in the network. Let SP_k, be the site which holds the primary copy of O_k, i.e., the only copy in the network that cannot be deallocated, hence referred to as primary site of the kth object. Each primary site SP_k, contains information about the whole replication scheme R_k of O_k. This can be done by maintaining a list of the sites that the kth object is replicated at, called from now on the *replicators* of O_k. Moreover, every site $S^{(i)}$ stores a two-field record for each object. The first field, is the primary site SP_k of it and the second the "nearest" site $SN_k^{(i)}$ of site i, which holds a replica of object k. In other words, $SN_k^{(i)}$ is the site for which the reads from $S^{(i)}$ for O_k, if served there, would incur the minimum possible communication cost. It is possible that $SN_k^{(i)} = S^{(i)}$, if $S^{(i)}$ is a replicator or the primary site of O_k. Another possibility is that $SN_k^{(i)} = SP_k$, if the primary site is the closest one holding a replica of O_k.

When a site $S^{(i)}$ reads an object, it does so by addressing the request to the corresponding $SN_k^{(i)}$. For the updates we assume that every site can update every object. This assumption, does not affect the ability to use our model in practical cases, where we have to enforce some privilege checking. Updates of an object O_k are performed, by sending the updated version to its primary site SP_k, which afterwards broadcasts it to every site in its replication scheme, R_k. The simplicity of this policy, allows us to develop a general cost model in Section 2.2 that can be used with minor changes to formalize various replication and consistency strategies.

2.1. The object transfer cost model

We are interested in minimizing the total network transfer cost (NTC), due to object movement, since the communication cost of control messages has minor impact to the overall performance of the system. There are two components affecting NTC. First is the NTC created from the read requests.

Let $R_k^{(i)}$ denote the total NTC, due to $S^{(i)}$'s reading requests for object O_k, addressed to the "nearest" site $SN_k^{(i)}$. This cost is given by the following equation:

$$R_k^{(i)} = r_k^{(i)} O_k C(i, SN_k^{(i)}), \tag{1}$$

where $SN_k^{(i)} = \{Site\ j \mid j \in R_k \wedge \min C(i, j)\}$.

The second component of NTC is the cost due to writes. Let $W_k^{(i)}$ be the total NTC, due to $S^{(i)}$s writing requests for object O_k, addressed to the primary site SP_k. $W_k^{(i)}$ is comprised of the cost occurring when $S^{(i)}$ sends the updated object to SP_k and the cost due to SP_k broadcasting the updates to the rest of the replicators as shown in the following equation:

$$W_k^{(i)} = w_k^{(i)} O_k \left[C(i, SP_k) + \sum_{\substack{\forall (j \in R_k) \\ j \neq 1}} C(SP_k, j) \right]. \tag{2}$$

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, in order to augment efficiency. Modelling such policies can also be done using our framework, e.g., by defining separately the object size, mean read size and the mean update size.

The cumulative NTC, denoted as D, due to reads and writes is given by

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

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. The total NTC is now refined as

$$D = \sum_{i=1}^{M} \sum_{k=1}^{N} \left\{ (1 - X_{ik}) [r_k^{(i)} O_k \min \{C(i, j) \mid X_{jk} = 1\} + w_k^{(i)} O_k C(i, SP_k) + X_{ik} \left(\sum_{x=1}^{M} w_k^{(x)} \right) O_k C(i, SP_k) \right\}. \tag{4}$$

Sites which are not replicators of object O_k create NTC equal to the communication cost of their reads from the "nearest" replicator, plus that of sending their writes to the primary site of O_k. Sites belonging to the replication scheme of O_k are only associated with the cost of sending/receiving all the updated versions of it, since reads are performed locally. Using the above formulation DRP is defined as

1. Find the assignment of 0, 1 values at the X matrix, that minimizes D.
2. Subject to the storage capacity constraint:

$$\sum_{k=1}^{N} X_{ik} O_k \leq s^{(i)} \quad \forall (1 \leq i \leq M).$$

3. Subject to the primary copies policy

$$X_{SP_k k} = 1 \quad \forall (1 \leq k \leq N).$$

2.2. Proof of NP-completeness

The DRP as presented above is a constrained optimization problem. The equivalent decision problem can be stated as

Given a network instance, with only primary copies existing (no replicas) and an integer K, is there an assignment of 0, 1 values to X matrix such as $D \leq K$ and the storage capacity constraint is satisfied?

It is easy to see that a non-deterministic algorithm, which chooses 0,1 assignment values row by row, while checking the storage constraint after completing each row of X, can decide the problem in polynomial time $O(MN)$. Thus, DRP belongs to NP.

In order to prove that it is also complete in NP, we will reduce it to the *Knapsack Problem* [26] the general statement of which is known to be NP-complete and is presented below:

Instance: Finite set U, for each $u \in U$ a size $s(u)$, a value $v(u)$, B, K', all positive integers.

Question: Is there a subset $U' \in U$ such that $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \geq K'$?

For every instance of the *Knapsack* problem, we can build a network so that a solution to the *Knapsack* would also be a solution to a DRP instance. The network is built as follows. We use a one by one mapping of objects belonging to U to the objects of the network (to do so we order the set U), i.e., $(N = |U|) \wedge (O_k = s(k) \forall 1 \leq k \leq N)$. We pick up a random number of sites $M - 1 (M \geq 2)$ and allocate all the network objects to each one of them assigning also $\sum_{u \in U} s(u)$ storage capacity, i.e.,

$$X_{ik} = 1 \wedge s^{(i)} = \sum_{u \in U} s(u)\ \forall (1 \leq i \leq M - 1 \wedge 1 \leq k \leq N).$$

Then, we randomly decide which of the $M - 1$ copies of an object k would be the primary one. Having defined all the primary copies, we add another "empty" site with storage capacity B, hence referred to as *Knapsack site Kns* because an assignment of objects to it that solves *Knapsack* would also solve DRP. We set $C(i, j) = 1\ \forall (1 \leq i, j \leq M)$, $W_k^{(i)} = 0\ \forall (1 \leq i \leq M \wedge 1 \leq k \leq N)$ and set

$r_k^{(Kns)} = v(k) \prod_{u \in U} s(u)/s(k) \ \forall (k \in U)$[1]. The NTC occurred due to the requests from all the other sites apart from *Kns* is zero, since the read requests are satisfied locally and no updates exist. Every object allocation scheme for *Kns* that minimizes NTC is a solution to DRP since the NTC of the rest part of the network remains zero. All link costs are equal to 1, so each object allocated in *Kns* reduces the total NTC by $r_k^{(Kns)} O_k$ or $v(k) \prod_{u \in U} s(u)$ regardless of the already allocated ones. Let U'' be the set of objects allocated to *Kns*. Let *Dinitial* denote the NTC occurred when each of $M-1$ sites contain all the objects and *Kns* is empty and *Dkns* denote the NTC occurred after completing object allocation in *Kns*. Then, the initial DRP decision problem can be restated as: Assuming the network instance which yields NTC equal to *Dinitial*, is there a subset U'' of the set of objects that if allocated at *Kns* site with respect to its capacity, the total NTC D will be $D \leq K$, where K can be any integer?

The inequality $D \leq K$ can be restated as: $Dinitial - \prod_{k=1}^{N} O_k \sum_{x \in U''} v(x) \leq K$, or $C \sum_{x \in U''} v(x) \geq E$ or finally as: $\sum_{x \in U''} v(x) \geq K''$ (where K, K'', E are any integers and C is constant). In order to answer the question whether a U'', such as $\sum_{x \in U''} v(x) \geq K''$ exists or not, we must answer the relevant Knapsack question ($\sum_{u \in U'} v(u) \geq K'$). Therefore, for every Knapsack instance we can construct a network as above, so that a solution to the Knapsack would also be a solution to the DRP thus, the DRP is reducable to the Knapsack and since it also belongs to NP it is NP-complete.

3. Static allocation

In this section we describe the simple replication algorithm based on the greedy method (SRA) and the genetic replication algorithm GRA. Both algorithms define a replication scheme for the objects assuming that no replicas exist and read/write frequencies are known and remain static in nature.

3.1. Data replication using a greedy algorithm

For each site $S^{(i)}$ and object O_k we define the *replication benefit* value $B_k^{(i)}$, as follows:

$$B_k^{(i)} = \frac{R_k^{(i)} - \left(\sum_{x=1}^{M} w_k^{(x)} O_k C(i, SP_k) - W_k^{(i)}\right)}{O_k}. \quad (5)$$

The above value represents the expected benefit in NTC terms, if we replicated O_k at $S^{(i)}$. This benefit is computed by using the difference between the NTC occurred from the current read requests and the NTC arising due to the updates to that replica amortized to the object size. Negative values of $B_k^{(i)}$ mean that replicating kth object, is inefficient from the "local view" of ith site. This does not necessarily mean that we are not able to reduce the total NTC by creating such a replica, but that the local NTC observed from the ith site will be increased.

To present our algorithm, we maintain a list $L^{(i)}$ for $S^{(i)}$ containing all the objects that can be replicated. An object O_k can be replicated at $S^{(i)}$ only if the remaining storage capacity $b^{(i)}$ of the site is greater than its size and the benefit value is positive. We also keep a list *LS* containing all the sites that have the "opportunity" to replicate an object. In other words, a site $S^{(i)} \in LS$ if and only if $L^{(i)} \neq \emptyset$. The SRA Algorithm performs in steps. In each step a site $S^{(i)}$ is chosen from *LS* in a round-robin fashion and the benefit values of all objects belonging to $L^{(i)}$ are computed. The one with the highest benefit is replicated and the lists *LS*, $L^{(i)}$ together with the corresponding nearest site value $SN_k^{(i)}$, are updated accordingly. The SRA algorithm is outlined as follows:

The SRA Algorithm:

(1) Initialize LS and all $L^{(i)}$.
(2) WHILE $LS \neq \emptyset$ DO
(3) $BMAX = 0, OMAX =$ NULL . /*$BMAX$ holds the current max $B_k^{(i)}$ value. $OMAX$ holds the identity of the Object for which $B_k^{(i)}$=*$BMAX$*/
(4) Pick up a site $S^{(i)} \in LS$ in a Round-Robin way.
(5) FOR each $O_k \in L^{(i)}$ DO
(6) Compute $B_k^{(i)}$.
(7) IF $BMAX \leq B_k^{(i)}$ THEN $BMAX = B_k^{(i)}, OMAX = k$
(8) ELSE IF ($B_k^{(i)} \leq 0$ OR $b^{(i)} < o_k$) THEN $L^{(i)} = L^{(i)} - \{O_k\}$.
(9) Replicate O_{OMAX}.
(10) $L^{(i)} = L^{(i)} - \{O_{OMAX}\}$. /*Remove $OMAX$ object from the list of potentials to be replicated*/
(11) FOR all sites in LS update the relevant $SN_{OMAX}^{(i)}$ field. /* Update "nearest sites"*/
(12) $b^{(i)} = b^{(i)} - o_k$. /*New remaining capacity*/
(13) IF $L^{(i)} = \emptyset$ THEN $LS = LS - \{S^{(i)}\}$ /*Remove $S^{(i)}$ if there are no other candidates (objects) to be replicated*/
(14) ENDWHILE

[1] By setting the reads as above we ensure they are integers. If non-integer reads were also allowed i.e. we had formalized DRP using *R/W* frequencies and not the actual number of requests, we could have simply used $r_k^{(Kns)} = v(k)/s(k)$.

The are MN iterations of the while-loop (2) in the worst case where each site has sufficient capacity to store all objects and the update ratios are low enough so as no object incurs negative benefit value. The time complexity for each execution of the while-loop is governed by the for-loop in (5) and the update of neighbors fields in (11) ($O(M+N)$ in total). Hence, we conclude that the complexity of our algorithm in the worst case is $O(M^2N + MN^2)$.

We presented SRA as a centralized algorithm. In its distributed version we assign $L^{(i)}$'s to their corresponding sites and LS to the network leader, which can be a monitor site. All the main calculations (5)–(10) are done locally, while (11) requires a broadcast of O_{OMAX} to all sites in order to update their $SN^{(i)}_{OMAX}$ field. The selection of $S^{(i)}$ in (4) is done by the leader who is responsible for notifying the sites accordingly.

It should be mentioned that since the algorithm replicates objects according to their "local" benefit value, it provides high solution quality, when independent read frequencies $r^{(i)}_k$ are significantly larger, compared to the total number of updates. This is better illustrated in our experiments at Section 5.

3.2. The evolutionary method

Genetic algorithms (GAs), introduced by Holland in 1975 [22], are search methods based on the evolutionary concept of natural mutation and the survival of the fittest individuals. Given a well-defined search space they apply three different genetic search operations, namely, selection, crossover, and mutation, to transform an initial population of chromosomes, with the objective to improve their quality. An outline of a generic GA is as follows.

Generate initial population.
Perform selection step.
while stopping criterion not met **do**
    Perform crossover step.
    Perform mutation step.
    Perform selection step.
end while.
Report the best chromosome as the final solution.

We demonstrate GRA's design in detail, by presenting our encoding mechanism and then the selection, crossover and mutation operators. Some additional notation related to GA is summarized in Table 2.

A chromosome encoding a replication scheme is a bitstring consisting of M genes (one for each site). Every gene is composed of N bits (one for each object). A 1 value in the kth bit of the ith gene, denotes that ith site holds a replica of kth object, otherwise it is 0. Using this encoding the total length of a chromosome is MN bits. The following scheme

Table 2

Symbol	Meaning
D	Total data transfer cost function
D_{prime}	Total data transfer cost function, of the initial alloc. scheme
f	Fitness value
$\dot{f}$	Average fitness value of the population
N_{p}	Population size (number of chromosomes)
N_{g}	Total number of generations
μ_{c}	Crossover rate
μ_{m}	Mutation rate

explains the above: Example chromosome

Site 1 Site M
| 101...0 | 100...1 |.....|001...1|
1... N Objects

A gene (site) is valid if and only if the total storage cost of allocating the required objects (1's in the gene) does not exceed the site's capacity; otherwise the gene is invalid. We also define a chromosome to be valid/invalid according to the existence of an invalid gene.

Fitness value f: The quality of each chromosome is measured by computing its fitness value. Our objective function D, defined in Section 2.2, helps us define f. In order to maintain uniformity over various problem domains, we normalize the fitness value to a convenient range of 0 to 1. The above need excludes us from picking up directly $f = D$, since D can take arbitrary high values. For our algorithm we consider our initial allocation scheme—an object appears in the network only at its primary site, that is $R_k = \{SP_k\} \forall (1 \leq k \leq N)$—and the NTC occurred in it, denoted by D_{prime}. It is reasonable to expect that every replication scheme invokes NTC D such that $D < D_{\text{prime}}$, meaning that from our starting allocation scheme with no replicas, there is still a lot of NTC to be saved by using replication of data. Considering the above, the definition of the normalized fitness value is straightforward:

$$f = \frac{D_{\text{prime}} - D}{D_{\text{prime}}}.$$

In the rare case that $f < 0$, we reset chromosome's fitness value to be 0 by copying the initial allocation scheme in it.

Generation of the initial population: We initialize half of the population by using the SRA algorithm, randomly picking up sites at its fourth step. The other half of the population is generated in a total random way. Moreover, half of the chromosomes produced by SRA are subjected to random perturbation of 1/4th of their values. The chromosome validity is checked and maintained throughout all the random decisions. The use of SRA together with random generation, results in the initial chromosomes being adequately homogeneous in their fitness values, while at the same time diverse enough in their building blocks (substrings). Furthermore, the average population fitness is high, even when

due to increased updates SRA fails to produce a good solution. SRA's complexity is $O(M^2N + MN^2)$ for M sites and N objects. Therefore, the total initialization time required is $O(N_pM^2N + N_pMN^2)$, where N_p stands for the population size,

Crossover/mutation/selection: We selected a two-point crossover mechanism to include in GRA. After the pairing of chromosomes, two bits are randomly selected and either the portion of the bit-string in between them or the two fractions not included by them, are swapped. The decision of which parts to juxtapose is random. The whole operation is performed with probability μ_c, known as crossover rate and may result in producing invalid chromosomes. Clearly, if this is the case, the only possible invalid genes, are the two (or one) containing the crossover points, since completely interchanging valid genes, cannot result in an invalid chromosome. To repair a gene validity violation, we exchange the parts of it that were not previously swapped.

Mutation is performed by simply flipping every bit with a certain probability μ_m, known as the mutation rate. Although mutation is not the primary search operation and some algorithms omit it, it is very useful in our design. The reason is that in order for our algorithm to have practical applications it should achieve good performance when tackling very large chromosomes (our test case in many experiments is 80 *sites and* 400 *objects*). A two point-crossover alone would not be able to explore the solution space fast while a multiple point crossover would result in increased constraint violations. Moreover, multiple point crossover is known to be highly disruptive [18]. Mutation aids in both the exploration and the exploitation of our solution space. Changing a bit value can result in violating the storage constraint, as well as deallocating an object from its primary site. In such case mutation of the specific bit is not performed.

Selection constitutes of two parts: evaluation of a chromosome and offspring allocation. Evaluation is performed by measuring its fitness value *f*, which depicts the quality of solution the chromosome represents. Offspring allocation is done by using the proportionate scheme (simple genetic algorithm [22]). This scheme allocates to the *i*th chromosome, $f_i/\bar{f}$ offspring for the next generation. SGA implements this scheme by using the roulette wheel selection, i.e., allocating a sector of the wheel equaling $2\pi f_i/\bar{f}$ to the *i*th chromosome and then create an offspring if a generated number in the range of 0 to 2π, falls inside the assigned sector of the string. The chromosomes under evaluation are N_p exactly, since after applying crossover and mutation the resulting chromosomes are immediately copied back in the place of their fathers. Instead of following this approach that can lead to large sampling errors, we selected the stochastic remainder technique [18] to incorporate in GRA. Following this method, a chromosome is assigned offspring according to the integer part of the proportionate fitness value in a deterministic way and the fractional parts are put in a roulette wheel, in order for the remaining offspring to be defined. Moreover, instead of evaluating N_p chromosomes (Simple Selection), we used the $(\mu + \lambda)$ Selection borrowed from evolutionary strategies [31]. Under this strategy from the initial population of $\mu = N_p$ size, two more subpopulations are created of total size λ, one from crossover and the other from mutation operator. The chromosomes of all these three populations $(\mu + \lambda)$ compete for the μ slots of the next generation. This choice although resulting in higher execution time (needs up to three times more fitness evaluations), was necessary because due to the hard constraint nature of the problem, applying mutation and crossover can result in worsening a generation. Finally, we implemented the *elitistic* approach, under which the best chromosome found until one generation before replaces the worst chromosome of the population. We allow the elite chromosome to be copied back once every five generations, in order to prevent premature convergence.

Large values of μ_c and μ_m, force a GA to explore the solution space, while low values favor exploitation. Optimal tuning of these, require extensive experimentations [18]. Typical values of the above parameters that guarantee good solution quality for some domains are: N_p={30, 100}, μ_c={0.9, 0.6}, μ_m = {0.01, 0.001}, respectively [14,19]. Of course, even with the best decisions on the above parameters, optimal solutions cannot be guaranteed, due to the algorithm's probabilistic nature. Unless otherwise stated the GRA's parameters after a series of experiments were fixed to: $N_p = 50$, $N_g = 80$, $\mu_m = 0.01$, $\mu_c = 0.9$.

As far as the time complexity of GRA is concerned, selection is clearly the hardest operation, because it involves the evaluation of fitness values, which in terms demands the computation of the objective function $D(O(M^2N))$. Thus, the complexity of GRA in the worst case is $O(N_gN_pM^2N + initialization)$.

4. Adaptive replication

Let R be the old replication scheme for the objects and R' the newly defined one. Let D_R and $D_{R'}$, represent the total NTC created by R and R', respectively, both computable using Eq. (4). Furthermore, let X'' be an $M \times N$ (0,1) matrix with its element X'_{ik} being 1 if O_k is replicated at $S^{(i)}$ under the R' scheme, and 0 otherwise. When realizing R' some replicas must be deleted and others must be created. We assume that replica creation is done by transferring a copy of the object from the respective primary site[2]. Assuming that replica deletion incurs no cost, the total NTC, denoted by for $I_{RR'}$, realizing R' is given by

$$I_{RR'} = \sum_{i=1}^{N}\sum_{k=1}^{M} X'_{ik}(1 - X_{ik})O_kC(i, SP_k), \tag{6}$$

[2] A more sophisticated policy would enable sites to get objects from any of the replicators. This gives rise to the scheduling problem of selecting replica deletions and creations so as to minimize the NTC. Tackling the above is out of the scope of this paper and is part of our future work.

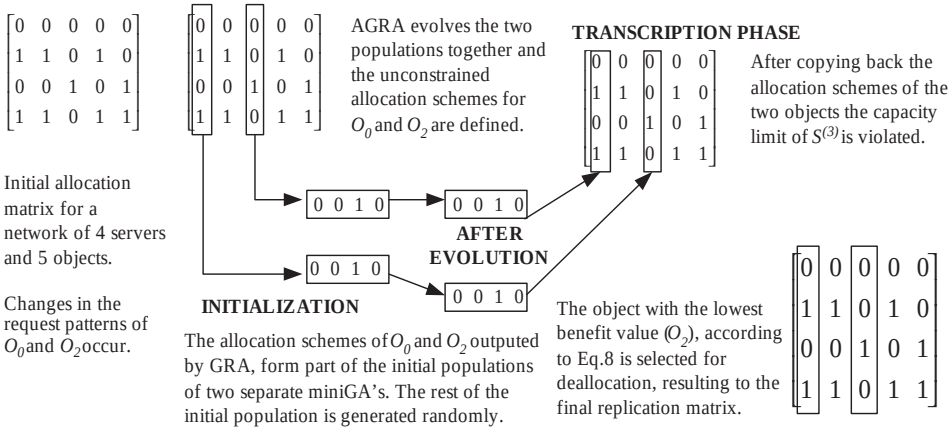


Fig. 1. Example of AGRA execution.

where X_{ik} is the allocation variable for the replication scheme R. The total benefit, denoted by $V_{RR'}$, for moving from the R' scheme to the R' is given by

$$V_{RR'} = D_R - (D_{R'} + I_{RR'}). \tag{7}$$

The adaptive data replication problem (ADRP) can be defined as: Given the X matrix find the values of X' that maximize $V_{RR'}$, subject to the storage capacity constraints. ADRP's scope is much different than DRP (Section 2). We use DRP to represent the allocation decisions made during nighttime by a monitor site that gathers statistics about object requests and takes decisions accordingly (we also made the indirect assumption that the actual cost for realizing a replication scheme during nighttime is insignificant and thus, can be omitted). On the other hand, ADRP is used to describe the situation when the replication scheme realized during nighttime, does not perform well during daytime, presumably because the exhibited request frequencies differ largely from the estimations used. For this purpose an algorithm that fine tunes, within reasonable time limits, the existing scheme instead of defining one from scratch is needed. Following, we present a genetic algorithm-based approach called AGRA.

4.1. The AGRA

Each chromosome in the population of AGRA is a bitstring of length M representing the assignments of O_k. AGRA takes as input the new patterns exhibited for O_k and computes a set of near optimal replication schemes for the specific object, without taking into account the storage capacity of the sites. Afterwards, the R_ks are incorporated to the initial GRA solutions, repairing any storage constraint violations by deallocating least beneficial objects. The modified population is then inserted to a micro-GRA and evolved for few generations in order to determine an even better solution. Fig. 1 illustrates the above.

The initialization of the first generation is performed in AGRA by randomly creating half of the population while the rest of it being obtained by solutions previously found from GRA. We make sure that the current replication scheme of O_k, always participates in the starting population. The three operations of GA (selection, crossover, mutation) come afterwards to define the population of the next generation. The fitness value of AGRA is given by

$$f_A = \frac{V_{RR'}}{D_R} = \frac{D_R - (D_{R'} + I_{RR'})}{D_R}.$$

In case $f_A < 0$, the chromosome's fitness value is set back to 0 by copying the initial R_k to the chromosome. Fortunately, by knowing D_R, f_A can be computed in $O(M^2)$ time through evaluating only the impact O_k has to the total NTC. The sampling space of AGRA is regular (as opposed to GRA where we used $(\mu+\lambda)$ selection) containing only the offspring and some part of the parents (those not subjected to crossover and mutation). Again, the stochastic remainder technique is used with the fractional parts being allocated in a roulette wheel. The rationale behind these choices as with all others concerning AGRA's design, is to reduce the running time. The number of generations A_g is set to 50 and the population size A_p to 10. The elitistic approach was followed as in GRA. Single point crossover was selected with equal probabilities of crossing the left and the right part of the chromosomes. Mutation means again simply flipping a bit. Constant crossover and mutation rates are used with values of 80% and 1%, respectively. After A_g generations, the GA part of the algorithm terminates having converged largely to high fitness solutions representing R_ks that distribute O_k to a degree that minimizes the NTC effects of R/W requests.

The best R_k found by AGRA is transcripted to half of the initial population of GRA, including the corresponding elite chromosome (current network replica distribution), while the remaining R_ks are randomly transcripted to the other half. Such transcription can result in storage capacity violation which needs to be resolved efficiently. Other than randomly deallocating objects until the constraint is satisfied, we followed a greedy method and calculate the negative impact each possible one object deallocation has in $V_{RR'}$.

Furthermore, instead of computing the actual performance difference which would require $O(M^2)$ time, we used the following estimation, computable in $O(M)$ time:

$$E_k^{(i)} = B_k^{(i)} / \left(\sum_{j=1}^{M} X'_{ik} \right). \quad (8)$$

$E_k^{(i)}$ depicts a site-object affinity measure. The object with the lowest value is evicted from $S^{(i)}$ free space. $B_k^{(i)}$ is the local benefit as defined in Eq. (5). The rationale behind amortizing the local benefit to the number of replicas existing in the network is to favor deallocations of objects for which many replicas exist. Having transcripted the R_ks to the initial solutions there are two valid options: (a) we stop here and pick up the chromosome of highest fitness value to realize the corresponding total replication scheme, (b) use the resulting population, as the initial population of a mini_GRA intended to run for small number of generations such as 5–10. We use and evaluate both policies.

Before concluding this section we would like to make some remarks on the usability of AGRA. The powerful technique of defining an optimal unconstrained replica distribution for a single object and transcript the solution to the total replication scheme, (repairing any capacity violations) can be incorporated in GRA's design as a separate operator, along with crossover, mutation and selection. Indeed, from the performance of the AGRA+5/10 GRA (Section 5), we expect that the performance of GRA would be considerably improved, since this operator forces GRA to search more promising regions of the solution space. We chose though to present this technique as a separate algorithm since it seemed reasonable to differentiate the static design of GRA which involved high running time from a method that is fast and can achieve good solution quality even when running without the mini-GRA.

5. Experimental results

Here, we present the results of our experiments carried out on a 200 MHz Ultra Sparc 2 machine. Two major experiments were conducted, one to evaluate the static algorithms and one the adaptive method. Comparing the performance of the static algorithms to the optimal solution obtained by exhaustive search, would have been the best way to illustrate the merits of our approaches. Exhaustive search though, was able to provide the optimum within reasonable running time, only for very small network instances (typically five sites six objects). Since such problem sizes have little practical meaning we followed another approach. We decided to use the primal dual method for (LP) [36] as implemented in MATLAB [37], in order to find an upper bound for the optimal solution. Some object allocations in the LP solution are fractional. By cutting off these allocations (LIP Linear Integer Programming) we obtained a lower bound for the optimal. The experimental results show that our algorithm performance fall in most cases between these two bounds. Even as we used the above technique we were able to obtain results for only medium sized networks (typically 15 sites 40–80 objects). For this reason, we also conducted a separate experiment series with large network instances and compare the relative performance of the algorithms. The solution quality in all cases, is measured according to the NTC percentage that is saved under the replication scheme found by the algorithms, compared to the initial one, i.e., when only primary copies exist. Finally, in the last experimental set we evaluated the performance of AGRA both as a standalone algorithm and in combination with the micro-GRA.

5.1. Workload

We generate the network structures in the following manner. The link costs were uniformly distributed between 1 and 10. This effectively represents the number of hops a TCP/IP packet should make in order to reach its destination, a link cost measurement that is commonly used, see for example [20]. Objects were created so as to resemble a generic web workload [5,39], i.e., object sizes followed a pareto distribution and their popularity the Zipf law [38]. The minimum object size was 4 K. Primary sites were chosen randomly. The total number of requests considered for large network instances were 1 million, while for medium sized networks 100,000. Two distinct cases were considered as to request generation. In the first one, each site has the same probability of requesting an object, while in the second case object requests are normally distributed to the sites. This is done in order to test the performance of our algorithms at the presence of hot spots. In all the experiments, the basic capacity of sites (C%) is proportional to the total size of objects. In order to ensure the creation of sites with diverse enough storing capabilities, the actual capacity of a site was a random value between (C/2)% and (3C/2)%.

Evaluating AGRA requires altering the original R/W patterns. Half of the new requests were randomly assigned to sites, while for the other half the assignment followed a normal distribution. This is again done to simulate hot spots. For each network instance, 15 different networks were generated. In all the experiments we recorded the average quality of replication schemes obtained (% NTC saving), together with the average execution time of the algorithms and the average number of replicas created in those 15 runs.

5.2. Performance of SRA and GRA

First, we assess the performance of SRA and GRA as compared to LP, LIP and random search, by varying the number of sites and objects. We fixed the site capacity to $C = 30\%$ and the update ratio to $U = 5\%$. Figs. 2 and 3 show the results when requests are normally distributed across the sites. The first observation is that GRA outperforms SRA in

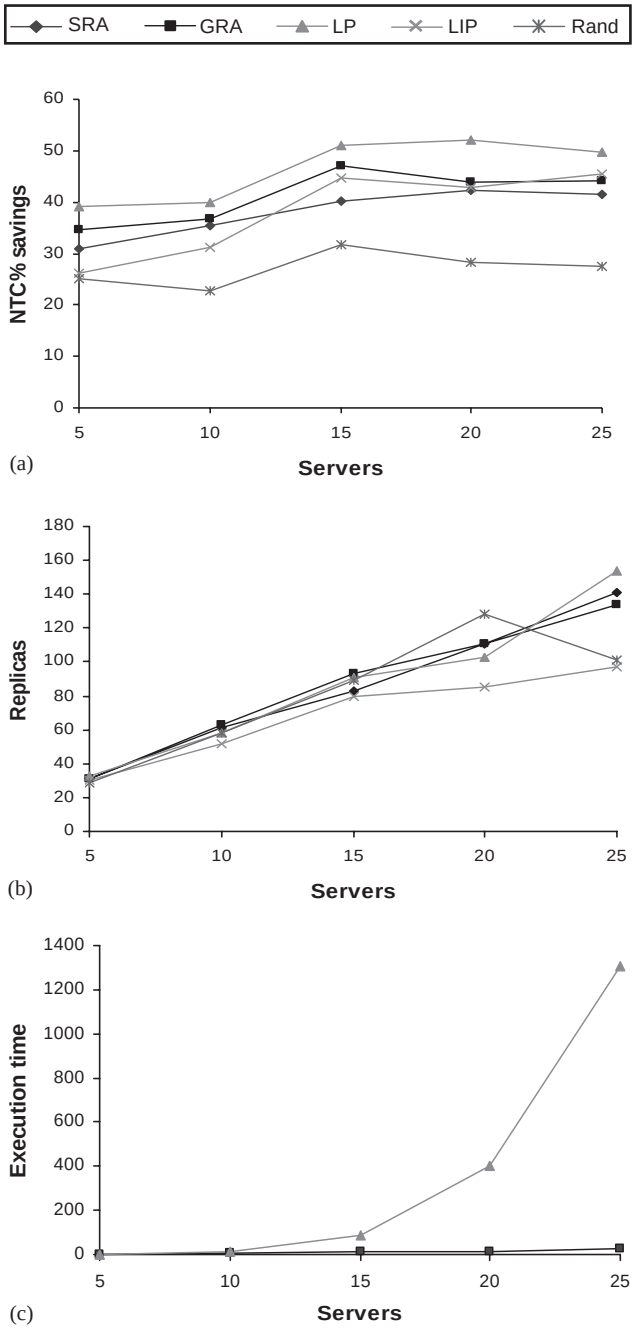


Fig. 2. (a) NTC% savings vs. sites ($N = 20$, $C = 30\%$, $U = 5\%$, normal), (b) replicas created vs. sites ($N = 20$, $C = 30\%$, $U = 5\%$, normal) and (c) execution time vs. sites ($N = 20$, $C = 30\%$, $U = 5\%$, normal).

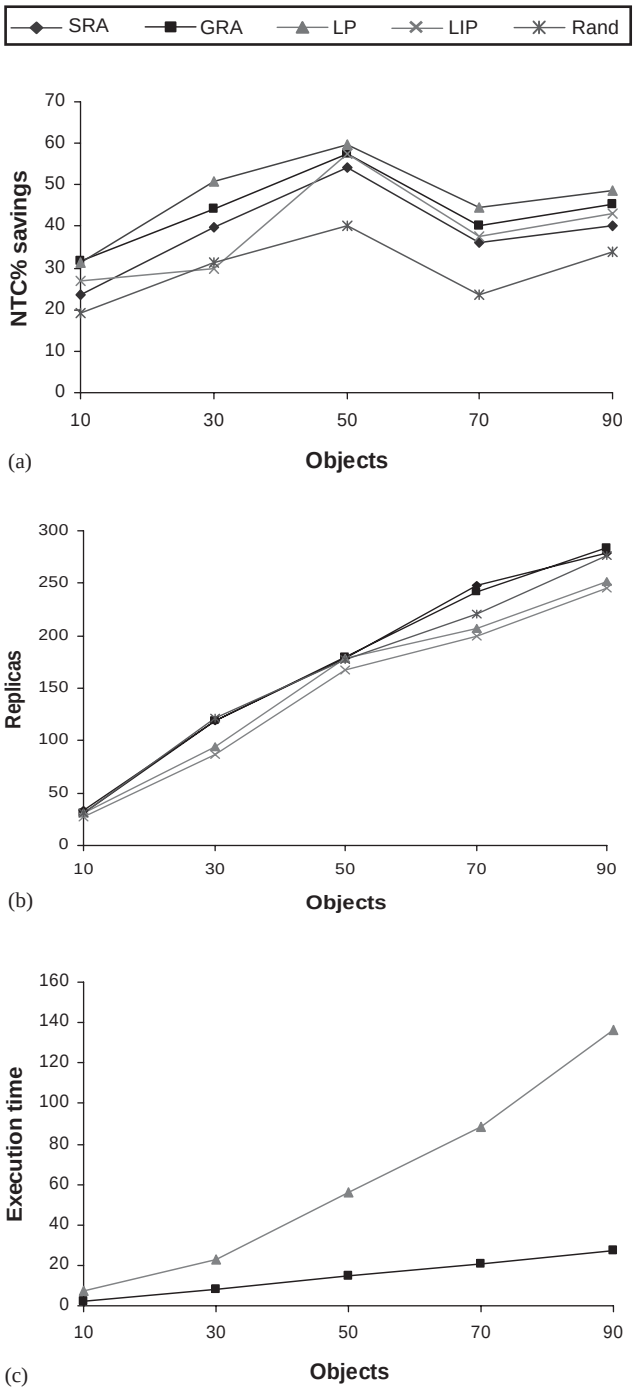


Fig. 3. (a) NTC% savings vs. objects ($M = 10$, $C = 30\%$, $U = 5\%$, normal), (b) replicas created vs. objects ($M = 10$, $C = 30\%$, $U = 5\%$, normal) and (c) execution time vs. objects ($M = 10$, $C = 30\%$, $U = 5\%$, normal).

terms of solution quality in all cases, while Random Search produces the lowest quality replication schemes among all the tested algorithms. The LP/LIP combination to approximate the global optimum is shown to be efficient since in most cases their performance difference is small (Figs. 4 and 5 further illustrate this). In the majority of problem instances GRA's performance falls between these two bounds while SRA does so in less cases. Fig. 2(a) shows an almost constant performance for the algorithms. The observation should be attributed to the fact that by adding a site in the network, we introduce additional traffic due to its local requests, together with more storage capacity to be used for replication. GRA explores and balances these diverse effects better than the greedy method. In Fig. 3(a) the performance of algorithms varies due to the fact that the generated network instances, had significantly different characteristics as

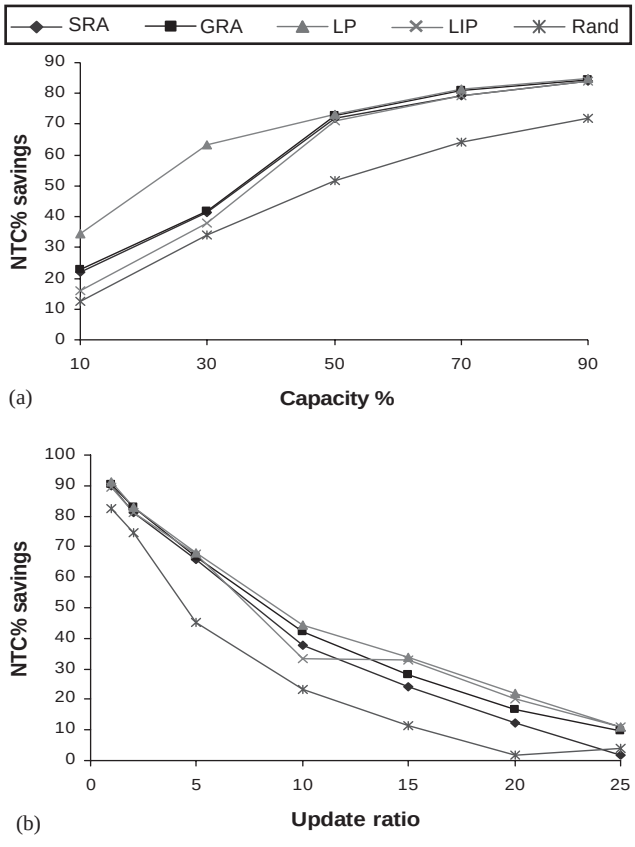


Fig. 4. (a) NTC% savings vs. capacity ($M=15$, $N=40$, $U=2\%$, normal) and (b) NTC% savings vs. update ratio ($M=15$, $N=40$, $C=80\%$, normal).

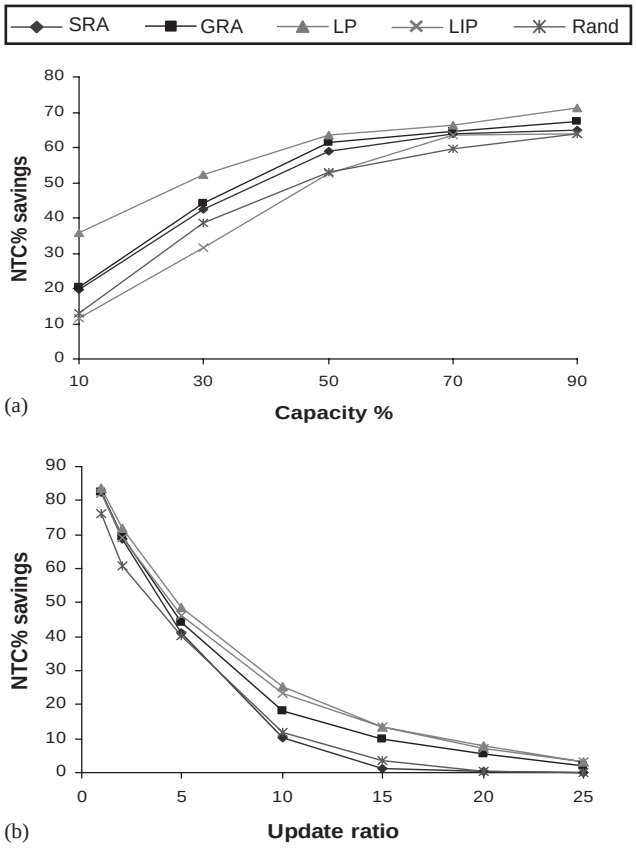


Fig. 5. (a) NTC% savings vs. capacity ($M=15$, $N=40$, $U=2\%$, uniform) and (b) NTC% savings vs. update ratio ($M=15$, $N=40$, $C=80\%$, uniform).

to the site capacities and request distributions among sites. Such diversibility on the generated instances has less impact as the network size increases (Figs. 6 and 7). Fig. 2(b) and 3(b) show an almost linear increase on the replicas created by all algorithms. Figs. 2(c) and 3(c) show the execution times for LP and GRA. LP's running time increases exponentially while GRA's running time is shown to be linear with a low growth rate, especially versus the number of sites (Fig. 2(c)). SRA (not shown in figure) required three orders of magnitude less running time compared to GRA, while LIP's running time is dominated by LP. Random Search's execution time was fixed to five times the one of GRA.

In the second set of experiments we investigated the impact, site capacity and update ratio have on the traffic savings obtained by the algorithms. We did so for two distinct cases, one for normal distribution of requests to the sites (Fig. 4) and one for uniform (Fig. 5). Creating replicas for an object that is already extensively replicated, is unlikely to result in significant traffic savings, since only a small portion of the sites will be affected and perform their reads with lower cost. It is also apparent that the update ratio sets an upper limit on the possible traffic reduction through replication, so that even with unlimited storage capacities at the sites, the replicate everything everywhere policy can be inadequate. Figs. 4(a) and 5(a) show that the cost savings tend to increase significantly at the beginning as the capacity increases, while after a certain point where the most read intensive objects are replicated, adding more storage space results to only marginal performance improvement. In the same figures we should note that GRA and SRA achieve comparable performance, primarily due to the small update ratio ($U=2\%$).

Figs. 4(b) and 5(b) depict the algorithms' performance as the update ratio increases. It is clear that all algorithms exhibit an exponential performance degradation, which is more apparent in the uniform case. Comparison between Figs. 4(b) and 5(b) shows that the solution quality difference between SRA and GRA increases to the update ratio, with the effect being more apparent when no hot spots exist in the network, Fig. 5(b). In order to understand why this happens, we should recall that SRA maintains a localized network perception. Increasing the updates, results in objects having decreased local benefit and forces SRA to take them out of the replication candidate list. In contrast, GRA with its randomized replica creation mechanism is able to select better replication schemes, especially in the uniform case where no super beneficial objects exist. The same plots also show that even when GRA's performance fails to be

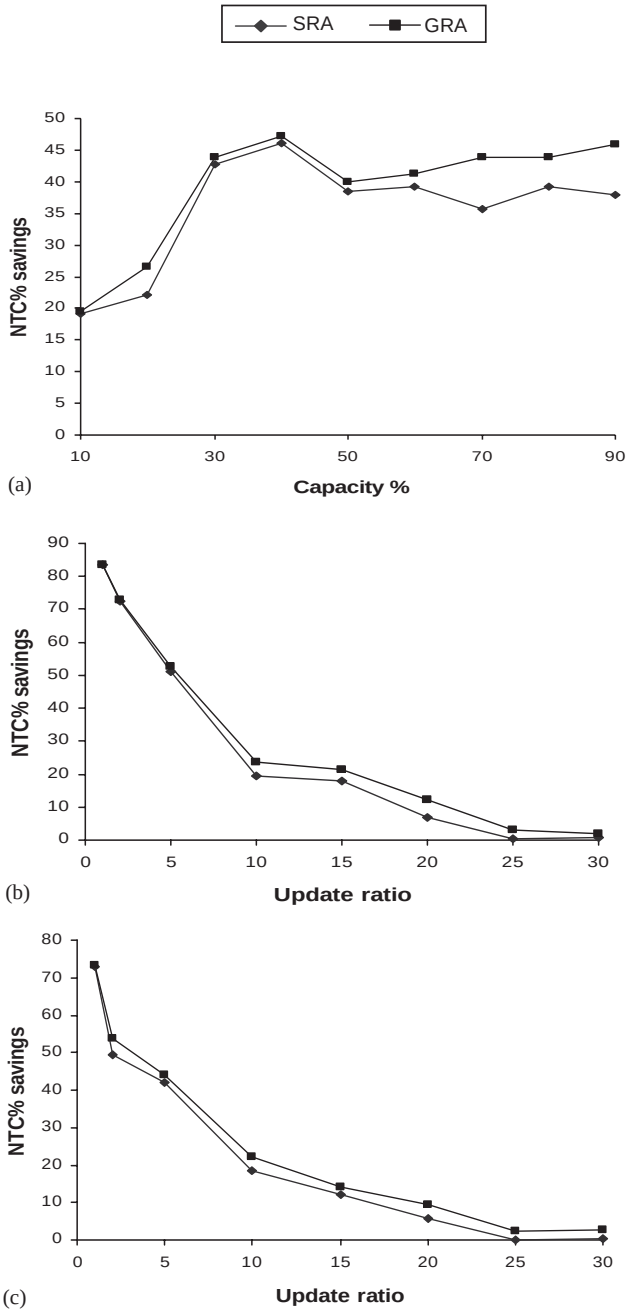


Fig. 6. (a): NTC% savings vs. capacity ($M = 30$, $N = 600$, $U = 5\%$, normal), (b) NTC% savings vs. update ratio ($M = 30$, $N = 600$, $C = 80\%$, normal) and (c) NTC% savings vs. update ratio ($M = 30$, $N = 600$, $C = 30\%$, normal).

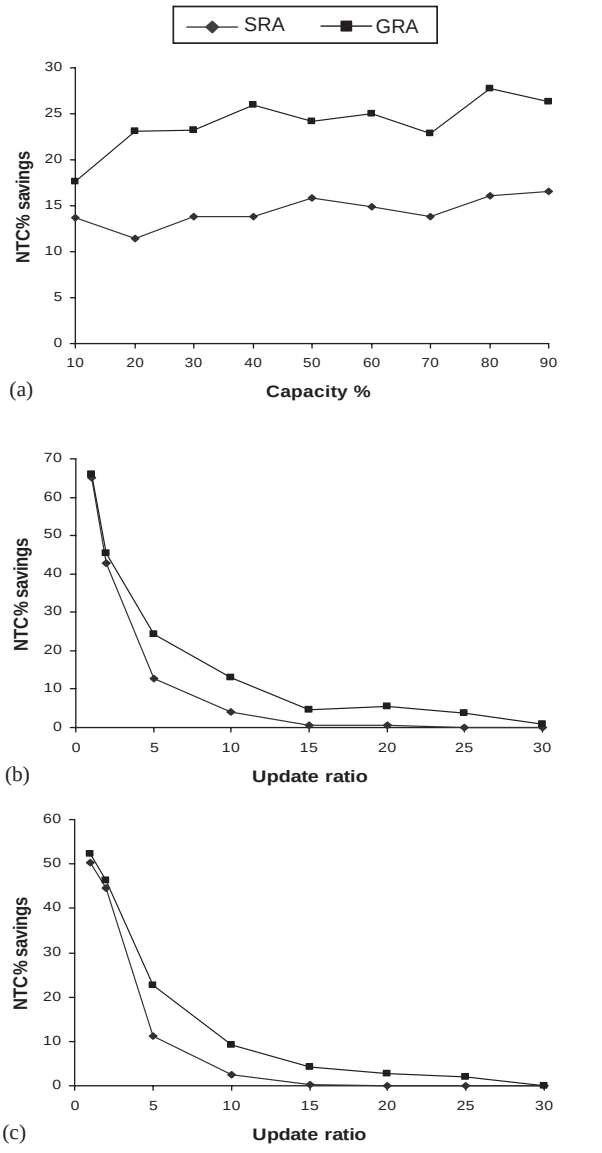


Fig. 7. (a) NTC% savings vs. capacity ($M = 30$, $N = 600$, $U = 5\%$, uniform, (b) NTC% savings vs. update ratio ($M = 30$, $N = 600$, $C = 80\%$, uniform) and (c) NTC% savings vs. update ratio ($M = 30$, $N = 600$, $C = 30\%$, uniform).

between LP and LIP[3], it is still very close to them, i.e., to the optimum.

From the first two series of experiments it became clear that GRA achieves solution quality close to the optimum in the majority of cases. It also became apparent that Random Search is unable to explore the problem space efficiently, while algorithms based on linear/integer programming will incur prohibitively high running time. For instance for a network consisting of 10 sites and 300 objects the resulting linear program will have 33,001 variables and 33,401 constraints, stretching the capabilities of any commercially available optimizer to its limits. Moreover, the running time explored in the paper referred to the linear programming relaxation of the problem (the one where fractional allocations are allowed), the actual (0,1) integer programming which is required, will most likely incur higher running time as additional constraints are added (variables should be either 0 or 1).

In the third class of experiments we investigated how SRA and GRA perform when the network size is large ($S = 30$, $O = 600$). Random Search gave inferior results to both

[3] For example, in Fig. 5(b) LP and LIP have negligible differences.

algorithms and is excluded from the plots. We were not able to obtain results from LP/LIP for this test case, due to the high running time of the algorithms. Figs. 6 and 7, illustrate the results both when one of the parameters determining the degree of replication (capacity/update ratio) is predominant in the network Fig. (6(a,b) and 7(a,b)) and when both constraints play significant role Fig. 6(c) and 7(c). From the figures it is clear that GRA achieves higher performance compared to SRA and that this performance difference is greater when the request distribution is uniform. The same trends exhibited in Figs. 4 and 5, are also present here, i.e., exponential decrease to the update ratio and an initial increase followed with an almost constant performance afterwards as storage space is added. We should notice here that Fig. 7(a) shows an almost constant performance for SRA, while GRA's performance increases in a rather linear way[4]. This is due to the fact that with $U = 5\%$ the point where further adding capacity results in only marginal savings is quickly reached[5].

Summarizing, GRA achieves more traffic savings (in many cases beyond 70%), than the greedy method and responds better to changes in the network size, the update ratios or the sites' capacities.

Furthermore, the quality of solutions obtained in a medium size network were close to optional. On the other hand, the greedy method apart from achieving good quality of solutions for small update ratios, runs in about orders of magnitude less time. Both algorithms were found to outperform random search. Linear programming was shown to incur high running time.

5.3. Performance of AGRA

Our test case is a network of $M = 30$, $N = 600$, $U = 2\%$, $C = 20\%$. To illustrate the main merits of AGRA we have considered the case where either reads or writes increase. The dual case of decrease is not included here but the results are equivalent. *Ch* denotes the percentage of rising in either reads or writes for an object that has changed its *R*/*W* pattern. *OCh* represents the percentage of objects in the network changing patterns and *R*, *U*, represent the percentage of changes being performed towards a read or write increase, respectively. So, for example in the network considered, $Ch = 600\%$, $OCh = 30\%$, $R = 80\%$ and $U = 20\%$, means that among the 600 total objects 144 have experienced an increase by 600% in their reads, while 36 a same increase in their write requests.

We consider various scenarios. Given a replication scheme (supposedly determined by a static algorithm), we evaluate this scheme according to the new read-write patterns and determine the current value (current legend label) of NTC savings. Then we run the AGRA algorithm (current+AGRA legend label) using the current scheme. We also run AGRA with five generations of mini_GRA (AGRA+5 GRA legend label) and then 10 generations of mini_GRA (AGRA+10 GRA legend label). Next, we run only GRA with 80 (current+80 GRA legend label). Finally, we run 80 generations of GRA (80 GRA legend label) but not with the current scheme, but with a population generated from scratch.

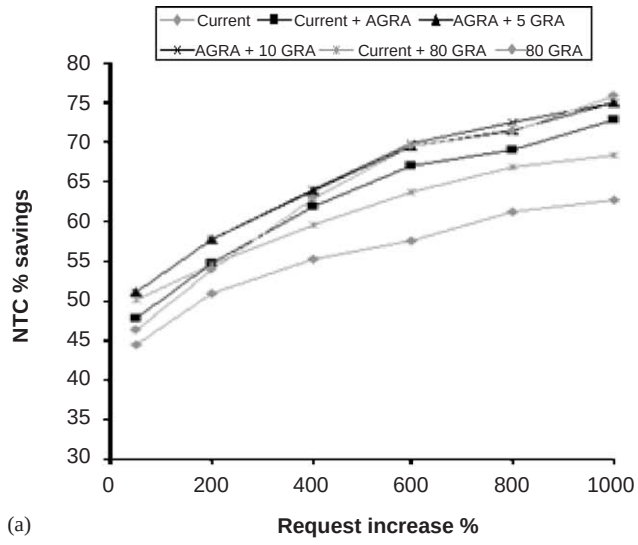
Fig. 8 show how the different policies perform as to the increase percentage (*Ch*). As it rises, the savings that GA policies achieve rise or drop depending on whether the increase refers to reads Fig. 8(a) or writes Fig. 8(c). The performance difference between AGRA and its combinations as compared to the static approaches is considerable (almost 15% in Fig. 8(c)). Moreover, if we do not have a dynamic method to adapt the replication scheme of the network, the existing scheme can be totally outdated and inefficient when the percentage of increase is significant, e.g., the 1000% case in Fig. 8(c). Fig. 9 shows similar trends when the increase percentage (*Ch*) is fixed and the percentage of changing objects (*OCh*) varies.

In Figs. 8(a) and 9(a), the performance of all policies when only reads are increased seems to converge to different upper bounds. This should be attributed to the fact that GA policies replicate intensively at the beginning, so as to exploit the available capacity to the maximum. After a certain point though, further replication is constrained due to storage limitations and thus, the savings tend to increase less rapidly. When only the number of updates increase, AGRA policies perceive an almost linear behavior Fig. 8(c) and 9(c) due to the fact that increasing the updates of a certain percentage of objects, only means that these objects should not be distributed widely. There are still though, enough read intensive objects which should be replicated and the deallocation criterion together with the unconstrained mini-GA, shift the replication scheme towards them.

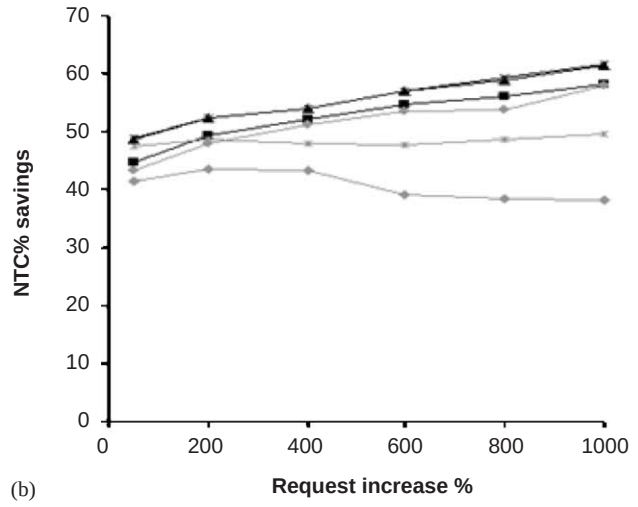
Fig. 10(a) and (b) show the performance of the algorithms as the pattern change shifts from 100% increase in updates towards 100% increase in reads. An interesting result is that among all the static methods, running GRA with a random initial population (80 GRA) is the best choice, for the cases were reads are increased while the (Current+80 GRA) policies are better when changes are performed in towards updates, or involve relatively low increase percentage (Fig. 8). The results obtained are encouraging towards the use of AGRA and its transcription/estimation method in dynamic environments. As we can see from all the above figures, AGRA's performance as stand-alone is significantly better than the current scheme, while its combination with the mini-GRA constantly outperforms all other static policies and is only worse by no more than 2% from (80 GRA) in the case were all changes are towards read increase. The (80 GRA) policy though, has prohibitively high execution

[4] The performance fluctuation in Fig. 6(a) and 7(a) and some paradoxes created due to it, should be attributed to variations between different network instances.

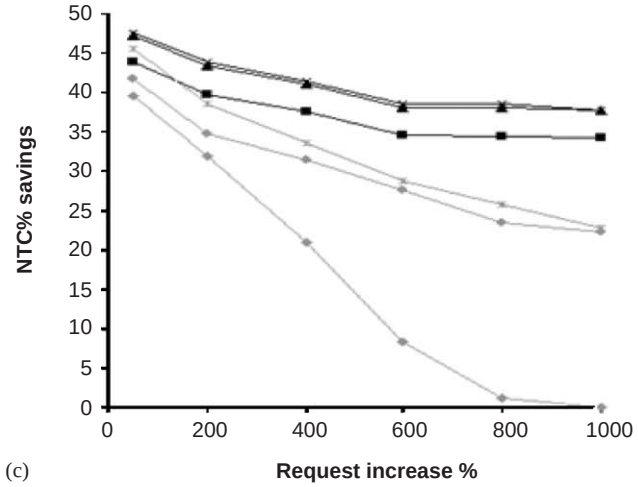
[5] Indeed, in further experiments with $U = 1$–3% the obtained plots followed the trends in Fig. 4(a) and 5(a).



(a)

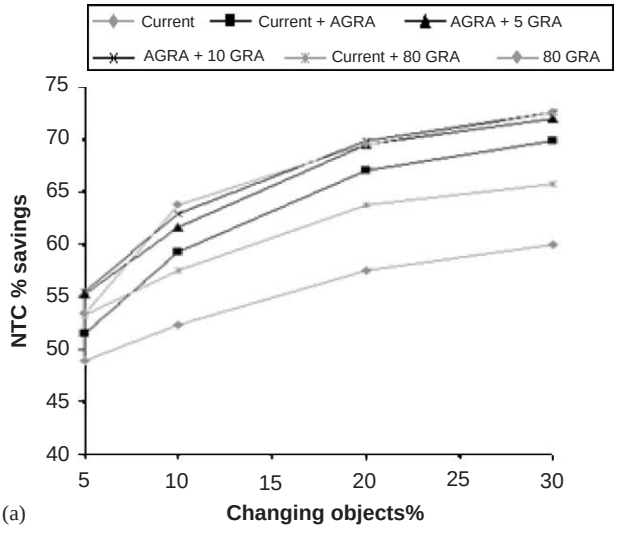


(b)

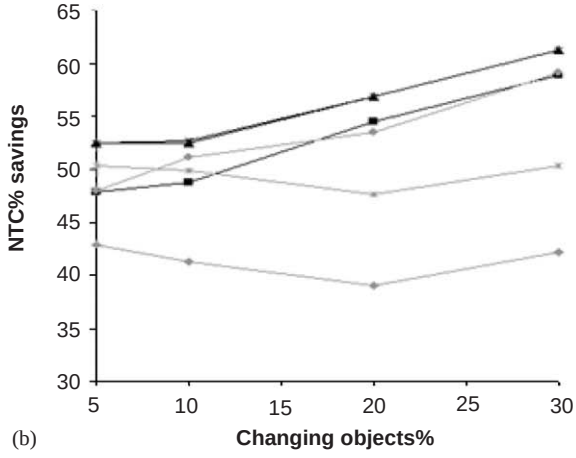


(c)

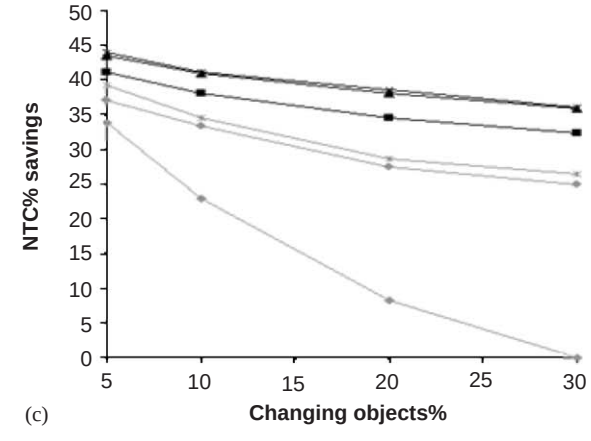
Fig. 8. (a) NTC% savings vs. Ch% ($OCh = 20\%$, $R/W = 100/0\%$), (b) NTC% savings vs. Ch% ($OCh = 20\%$, $R/W = 50/50\%$) and (c) NTC% savings vs. Ch% ($OCh = 20\%$, $R/W = 0/100\%$).



(a)



(b)



(c)

Fig. 9. (a) NTC% savings vs. OCh% ($Ch = 600\%$, $R/W = 100/0\%$), (b) NTC% savings vs. OCh% ($Ch = 600\%$, $R/W = 50/50\%$) and (c) NTC% savings vs. OCh% ($Ch = 600\%$, $R/W = 0/100\%$).

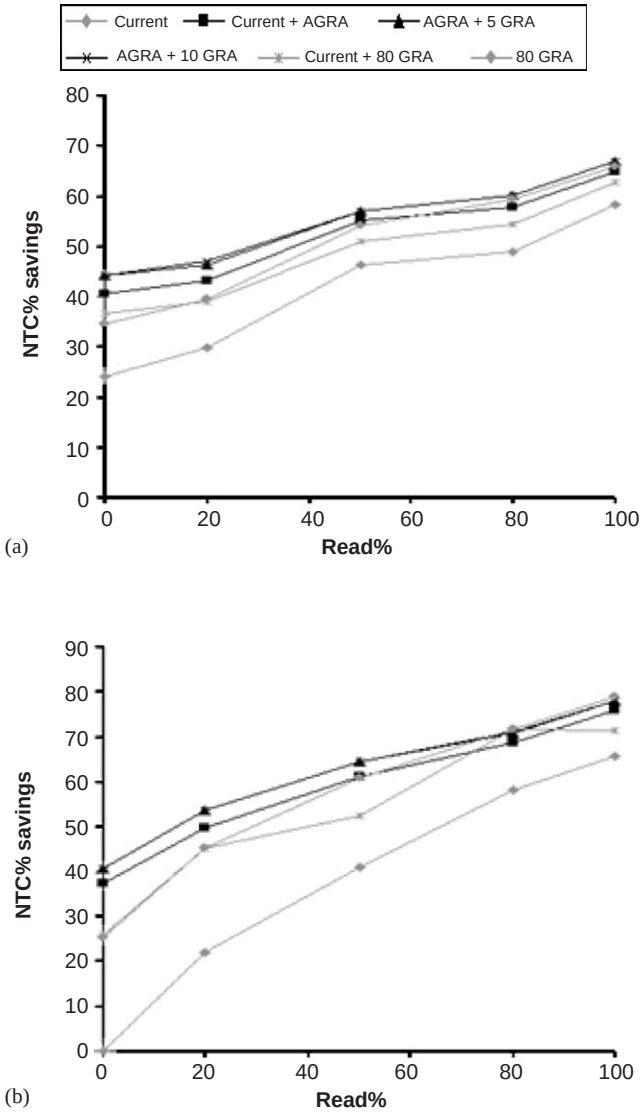


Fig. 10. (a) NTC% savings vs. $R/W\%$ ($Ch = 400\%$, $OCh = 20\%$), (b) NTC% savings vs. $R/W\%$ ($Ch = 1000\%$, $OCh = 20\%$).

time (around 1000 s for this network size), while AGRA as stand-alone is two orders of magnitude faster. Moreover, the number of changing objects have a marginal effect in the execution time of AGRA.

Summarizing the results of the experimental evaluation we conclude that when static patterns are considered the GRA algorithm promises good performance in expense of high running time. In dynamic environments though AGRA performs really well especially when combined with the mini-GRA. This is because the mini-GRA using the adaptive part of AGRA enhances the exploration power of our static design. As a result, the combination of the two algorithms proves to be very efficient in the very first five generations of mini-GRA, while the quality improvement with 10 generations or more is only marginal. The running time of AGRA with mini-GRA is acceptable for the requirements of a dynamic environment, while the quality of solutions obtained is if not higher at least comparable to the more time consuming static genetic algorithm method.

6. Related work

The data replication problem as presented in Section 2 is an extension of the classical file allocation problem (FAP). Chu [13] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [11] extended this work by distinguishing between updates and read file requests. Eswaran [16] proved that Casey's formulation was NP complete. In [25] Mahmoud et al. provide an iterative approach that achieves good solution quality when solving the FAP for infinite site capacities. A complete although old survey on the FAP can be found in [15]. Apers [4] considered the data allocation problem (DAP) in distributed databases where the query execution strategy influences allocation decisions. Bellatreche et al. [8] proposed an iterative approach to allocate rules and data in distributed deductive databases (rule allocation problem RAP), while Kwok et al. [33] proposed several algorithms to solve the data allocation problem in distributed multimedia databases (without replication), also called as video allocation problem (VAP).

Most of the research papers outlined in [15], aim at formalizing the problem as an optimization one, sometimes using multiple objective functions. Network traffic, throughput of servers and response time exhibited by users are considered for optimization. Although a lot of effort was devoted in providing comprehensive models, little attention was paid in proposing good heuristics to solve an often NP-hard problem. Furthermore access patterns are assumed to remain static and solutions in the dynamic case are obtained by reexecuting a perhaps time consuming linear programming technique.

Some on-going work is related to dynamic replication of objects in distributed systems when the read-write patterns are not known a priori. Awerbuch's et al. work in [6] is significant from a theoretical point of view, but the adopted strategy that before commuting an update replicas of the object must be deleted, can prove difficult to implement in a real-life environment. In [35] Wolfson et al. proposed an algorithm which leads to optimal single file replication in case of a tree network. The performance of the scheme for general network topologies is not clear though. Dynamic replication protocols were also considered under the Internet environment. Heddaya et al. [21], proposed a protocol that load balances the workload among replicas. It burdens, however, routers with keeping track of the replicas. In [29] Rabinovich et al. proposed a protocol for dynamically replicating the contents of an internet service provider ISP so as to improve the client-server proximity without overloading any of the servers. However, the update cost was not included, while the use of threshold values is likely to make the performance sensitive to their values.

Taking into account the more pragmatic scenario in today's distributed information environments, we decided to 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 site" policies. Recently genetic algorithms have been used for various optimization problems including multiprocessor scheduling [32], graph partitioning [12], task mapping [34] and multiprocessor document allocation [17]. We take advantage of their capability to explore fast and efficient the solution space of a problem in order to design static and adaptive algorithms for data replication. The main merits of using a genetic algorithm approach in the dynamic case lies in the proposed adaptive GA that uses *existing* knowledge about replica distribution in order to quickly define a new scheme.

7. Conclusions

In this paper, we addressed the data replication problem and developed a cost model, which is applicable to large distributed systems. We proposed a greedy algorithm to solve the problem. Having obtained initial solutions from our greedy approach, we designed a genetic algorithm. We evaluated both approaches and assessed the trade-off between running time/solution quality. Experimental analysis illustrated that the GA design constantly outperforms the greedy method in terms of solution quality. On the other hand SRA is much faster than GRA. Moreover, for small and medium sized networks SRA's performance is comparable to that of GRA. However, for an environment where static algorithms are less than useful, we propose AGRA which adapts to the changing environment very quickly and readjusts the replication scheme with solutions that are comparable to static algorithms. Therefore, AGRA combined with the mini_GRA is the ultimate choice in a dynamic environment.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, E. Fox, Caching Proxies: Limitations and Potentials, Electronics Proceedings of the Fourth World Wide Web Conf'95: The Web Revolution, Boston MA, December 11–14, 1995.
[2] T. Anderson, Y. Breitbart, H.F. Korth, A. Wool, Replication, consistency and practicality: are these mutually exclusive?, ACM SIGMOD'98, Seattle, June 1998.
[3] D. Agrawal, A.J. Bernstein, A nonblocking quorum consensus protocol for replicated data, IEEE Trans. Parallel Distributed Systems 2(2), (April 1991), 171–179.
[4] P.M.G. Apers, Data allocation in distributed database systems, ACM Trans. Database Systems 13(3), (September 1988), 263–304.
[5] M.F. Arlitt, C.L. Williamson, Internet Web Servers: Workload Characterization and Performance implications, IEEE/ACM Trans. Networking 5(5), (October 1997), 631–645.
[6] B. Awerbuch, Y. Bartal, A. Fiat, Optimally-competitive distributed file allocation, 25th Annual ACM STOC, Victoria, BC, Canada, 1993, pp. 164–173.
[7] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, P. Sturm, Enhancing the web infrastructure—from caching to replication, IEEE Internet Comput. (March–April 1997), 18–27.
[8] L. Bellatreche, K. Karlapalem, L. Qing, An Iterative Approach for Rules and Data Allocation in Distributed Deductive Database Systems, in Seventh International Conference on Information and Knowledge Management (ACM CIKM'98), Washington DC, November 1998, pp. 356–363.
[9] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, A. Secret, The world-wide web, Comm. Assoc. Comput. Mach. 37(8), (August 1993), 76–82.
[10] A. Bestavros, WWW traffic reduction and load balancing through server-based caching, IEEE Concurrency: Special Issue on Parallel and Distributed Technology, vol. 5, January–March 1997, pp. 56–67
[11] R.G. Casey, Allocation of copies of a file in an information network, Proceedings of the Spring Joint Computer Conference, IFIPS, 1972, pp. 617–625.
[12] R. Chandrasekharam, S. Subhramanian, S. Chaudhury, Genetic algorithm for node partitioning problem and applications in VLSI design, IEE Proc., 140(5), (September 1993), 255–260.
[13] W.W. Chu, Optimal file allocation in a multiple computer system, IEEE Trans. Comput. C-18 (10) (1969) 885–889.
[14] K.A. DeJong, W.M. Spears, An analysis of the interacting roles of population size and crossover in genetic algorithms, Proceedings of the First Workshop Parallel Problem Solving from Nature, Springer, Berlin 1990, pp. 38–47.
[15] L.W. Dowdy, D.V. Foster, Comparative models of the file assignment problem, ACM Comput. Surveys 14(2) (June 1982), 287–313.
[16] K.P. Eswaran, Placement of records in a file and file allocation in a computer network, Inform. Process. (1974) 304–307.
[17] O. Frieder, H.T. Siegelmann, Multiprocessor document allocation: a genetic algorithm approach, IEEE Trans. Knowledge Data Eng. 9(4), (July/August 1997), 640–642.
[18] D.E. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley, Reading, MA, 1989.
[19] J.J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Trans. Systems Man and Cybernetics SMC-16(1), (January/February 1986), 122–128.
[20] J.S. Gwertzman, M. Seltzer, The case for geographical push-caching, Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V), IEEE Computer Society Press, Los Alamitos, CA., 1995, pp. 51–55.
[21] A. Heddaya and S. Mirdad, WebWave: globally load balanced fully distributed caching of hot published documents, in Proceedings of the 17th International Conference on Distributed Computing Systems.
[22] J.H. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, MI, 1975.
[23] T. Loukopoulos, I. Ahmad, Replicating the contents of a WWW Multimedia Repository to Minimize Download Time, IPDPS'00, Cancun, Mexico, May 2000.
[24] T. Loukopoulos, I. Ahmad, Static and adaptive data replication algorithms for fast information access in large distributed systems, IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000.
[25] S. Mahmoud, J.S. Riordon, Optimal allocation of resources in distributed information networks, ACM Trans. Database Systems 1(1) (March 1976), 66–78.
[26] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Interscience Series in Discrete Mathematics and Optimization, New York, 1990.
[27] J. Mogul, Network behavior of a busy Web server and its clients, Research Report 95/5, DEC Western Research, Palo Alto CA, 1995.
[28] Network Appliances' NetCache, White paper at: http://www.netapp.com/products/level3/netcache/webcache.html.
[29] M. Rabinovich, I. Rabinovich, R. Rajaraman, A. Aggarwal, A dynamic object replication and migration protocol for an Internet

hosting service, IEEE International Conference on Distributed Computing Systems, May 1999.
[30] M. Rabinovich, O. Spatschek, Web Caching and Replication, Addison-Wesley, 2002.
[31] H. Scwefel, Evolution and Optimum Seeking, Wiley, New York, 1994.
[32] Yu-Kwong Kwok, I. Ahmad, Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, J. Parallel Distributed Comput. 47(1), (November 1997), 58–77.
[33] Y.K. Kwok, K. Karlapalem, I. Ahmad, N.M. Pun, Design and Evaluation of data allocation algorithms for distributed database systems, IEEE J. Selected Areas Comm. (Special Issue on Distributed Multimedia Systems), 14(7) (September 1996) 1332–1348.
[34] L. Wang, H. J. Siegel, V.P. Roychowdhury, A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments, Proceedings of the Fifth Heterogeneous Computing Workshop, 1996, pp. 72–85.
[35] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, ACM Trans. Database Systems 22(4), 255–314 (June 1997).
[36] S.P. Bradley, A.C. Hox, T.L. Magnanti, Applied Mathematical Programming, Addison-Wesley, Reading, MA, 1977.
[37] http://www.mathworks.com
[38] G.K. Zipf, Human Behavior and the Principle of Least-Effort, Addison-Wesley, Cambridge, MA, 1949.
[39] P. Barford, A. Bestavros, A. Bradley, M. Crovella, Changes in Web client access patterns: Characteristics and caching implications, WWW J 2 (1) (1999) 15–28.

Thanasis Loukopoulos received his Diploma in Computer Engineering and Informatics from theUniversity of Patras, Greece, in 1997. He was awarded a Ph.D. degree in Computer Science by the Hong Kong University of Science and Technology (HKUST) in 2002.
After receiving his Ph.D. he worked as a Visiting Scholar in HKUST. Currently, he is doing his military service. His research areas of interest include content distribution networks, P2P systems, and web services.

Ishfaq Ahmad received a Ph.D. degree in Computer Science from Syracuse University, New York, USA, in 1992. His recent research focus has been on developing distributed multimedia systems, video compression techniques, scheduling and mapping algorithms for scalable architectures, heterogeneous computing systems, and web management. His research work in these areas is published in over 150 technical papers in refereed journals and conferences, with best paper awards at Supercomputing 90 (New York), Supercomputing ‘91 (Albuquerque), and 2001 International Conference on Parallel Processing (Spain). He is currently a full professor of Computer Science and Engineering in the CSE Department of the University of Texas at Arlington. Prior to joining UT Arlington, he was an associate professor in the Computer Science Department at HKUST in Hong Kong. At HKUST, he was also the director of the Multimedia Technology Research Center, an officially recognized research center that he conceived and built from scratch. The center commercialized several of its technologies to its industrial partners worldwide. He has been on the program committee of more than 50 international conferences and is an Associate Editor of Cluster Computing, Journal of Parallel and Distributed Computing, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Concurrency, and IEEE Distributed Systems Online.