

Static and Adaptive Data Replication Algorithms for Fast Information Access in Large Distributed Systems

Thanasis Loukopoulos and Ishfaq Ahmad

Department of Computer Science
The Hong Kong University of Science and Technology, Hong Kong

Abstract

Creating replicas of frequently accessed objects across a read-intensive network can result in large bandwidth savings which, in turn, can lead to reduction in user response time. On the contrary, data replication in the presence of writes incurs extra cost due to multiple updates. The set of sites at which an object is replicated constitutes its replication scheme. Finding an optimal replication scheme that minimizes the amount of network traffic, given read and write frequencies for various objects, is NP-complete in general. We propose two heuristics to deal with this problem for static read and write patterns. The first is a simple and fast greedy heuristic that yields good solutions when the system is predominantly read-oriented. The second is a genetic algorithm that through an efficient exploration of the solution space provides better solutions for cases where the greedy heuristic does not perform well. We also propose an extended genetic algorithm that rapidly adapts to the dynamically changing characteristics such as the frequency of reads and writes for particular objects.

1 Introduction

Undesired delays in accessing the information have triggered much research activity on improving web performance through caching [1], [4], [7] (proxy servers) and replication [6] (mirror servers). In this paper we address the problem of replicating data objects according to their size, read and write frequencies as well as sites' capacities, in order to minimize the network traffic, which would lead to the reduction of average response time. A more spherical study of replication would include consistency and fault tolerance issues [2]. This work focus on clarifying the data replication problem. We provide a detailed formulation in the context of both read and write queries. The resulting Data Replication Problem (DRP) is provably NP-complete. We then propose a greedy heuristic, called (SRA), which yields good solutions when the number of reads is large. However, the greedy algorithm has its shortcoming in dealing with situations when the number of writes is large and the storage capacity is limited. To overcome these limitations, we propose an efficient genetic algorithm based heuristic called (GRA) that provides good quality solutions. Both SRA and GRA are static algorithms. To deal with the dynamic nature of the web we propose an extension of the GRA algorithm which adapts to the dynamically changing characteristics such as the number of reads and writes for particular objects. All algorithms are extensively evaluated.

The rest of the paper is organized as follows: Section 2 elaborates the problem and describes a cost model for the total data transfer cost. Section 3 describes the SRA algorithm. Section 4 describes the GRA and Section 5 presents the adaptive (AGRA). Section 6 and 7, respectively, include the experimental results and the related work. Section 8 includes some concluding remarks and future work.

2 The Data Replication Problem

First, we describe the inputs to the data replication problem (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 $\{O_1, O_2, \dots, O_N\}$. 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. Notation and their meanings

Symbol	Meaning
O_k	k th object
o_k	Size of object k
$S^{(i)}$	i th 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 k th object
$SN_k^{(i)}$	Nearest site of site i , which holds object k
R_k	Replication scheme of the k th object
D	Total data transfer cost function
$B_k^{(i)}$	Benefit value, that is, the NTC saves we can achieve by replicating the j th object at the i th site.

2.1 Replication Policy

Our 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 k th 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 where the k th object is replicated at, called from now on the *replicators* of O_k . Moreover, every site $S^{(i)}$ stores a two-field record for each object. The first field is its primary site SP_k 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. 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.

2.2 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:

$$R_k^{(i)} = r_k^{(i)} o_k C(i, SN_k^{(i)}) \quad (1) \quad \text{where } SN_k^{(i)} = \{Site \ j | j \in R_k \wedge \min C(i, j)\}$$

The second component of NTC is the cost arising due to the 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 . This cost is given by the following equation:

$$W_k^{(i)} = w_k^{(i)} o_k \left[C(i, SP_k) + \sum_{\substack{j \in R_k \\ j \neq i}} C(SP_k, j) \right] \quad (2)$$

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 (modelling such policies can also be done using our framework). The cumulative NTC, denoted as D , due to reads and writes is given by:

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

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

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

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 associated with the cost of sending/receiving all the updated versions of it. Using the above formulation, the Data Replication Problem (DRP) can be defined as:

Find the assignment of 0, 1 values in the X matrix that minimize D .

Subject to the storage capacity constraint:

$$\sum_{k=1}^N X_{ik} o_k \leq b^{(i)} \quad \forall (1 \leq i \leq M)$$

Subject to the primary copies policy:

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

The DRP as presented above is a constrained optimization problem. The equivalent decision problem is reducible to the *Knapsack Problem* [19] which is known to be NP-complete

3 A Greedy Method for Data Replication

In this section we describe the simple replication algorithm based on the greedy method. 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, which would be eliminated if we made a replica and the NTC arising due to the updates to that replica. Since we want to consider the benefit per storage data unit, we divide the difference of NTCs by the object size. Negative values of $B_k^{(i)}$ mean that replicating k th object is inefficient from the “local view” of i th 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 i th 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, i.e., $b^{(i)} \geq o_k$ 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:

- (1) Initialize LS and all $L^{(i)}$.
- (2) WHILE $LS \neq \emptyset$ DO
/* $BMAX$ holds the current max $B_k^{(i)}$ value. $OMAX$ holds the identity of the Object for which $B_k^{(i)} = BMAX$ */
- (3) $BMAX = 0$, $OMAX = NULL$.
- (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} .
- /*Remove $OMAX$ object from the list of potentials to be replicated*/
- (10) $L^{(i)} = L^{(i)} - \{O_{OMAX}\}$.
/* Update “nearest sites”*/
- (11) FOR all sites in LS update the $SN_{OMAX}^{(i)}$ field.
- (12) $b^{(i)} = b^{(i)} - o_k$. /*New remaining capacity*/
- /*Remove $S^{(i)}$ if there are no other candidates (objects) to be replicated*/
- (13) IF $L^{(i)} = \emptyset$ THEN $LS = LS - \{S^{(i)}\}$.
- (14) ENDWHILE

Each execution of the while-loop has complexity $O(M + N)$. In the worst case where each site has enough capacity to hold all the objects and the number of updates is zero, there are MN such iterations. Hence, we conclude that

the complexity of our algorithm 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. All the main calculations are done locally, while (11) requires a broadcast of O_{MAX} to all sites in order to update their $SN_{MAX}^{(i)}$ field. The selection of $S^{(i)}$ in (4) is done by the leader and followed by a token passing mechanism. The token is returned to the leader upon completion of (13). It should be mentioned that since the algorithm replicates objects according to their "local" benefit value, it provides good solution quality when independent read frequencies ($r_k^{(i)}$'s) are significantly larger than updates. This is better illustrated in our experiments at Section 6.

4 The Evolutionary Method

Genetic algorithms (GAs), introduced by Holland in 1975 [17], 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. Fundamental to the GA structure is the notion of chromosome, which is an encoded representation of a feasible solution, most commonly a bit string. Before the search process starts, a set of chromosomes is initialized to form the first generation. Then the three genetic search operations are repeatedly applied, in order to obtain a population with better characteristics. 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.

Encoding mechanism: A chromosome consists of M genes (one for each site). Every gene is composed of N bits (one for each object). A 1 value in the k th bit of the i th gene simply denotes that the i th site holds a replica of k th object. Using this encoding the total length of a chromosome is MN bits. The following scheme explains the above:

	Site 1	Site M
Example Chromosome:	101...0 100...1 001...1	
	1... N Objects	

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 need to normalize the fitness value to a convenient range of 0 to 1. For our algorithm we consider our initial allocation scheme, i.e., an object appears in the network only at its primary site, and the NTC occurred in it, denoted by D_{prime} and define the fitness value in the following way:

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

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

Validity of a chromosome: We define a gene (site) to be 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.

Generation of the initial Population: We initialize the population by using SRA algorithm N_p times, where N_p stands for the population size. In order to provide diversity, instead of picking up the "start-up" sites (step 4 of the algorithm) in a round-robin way, we do it randomly. Moreover, half of the population defined by SRA is subjected to random perturbation of 1/4th of their values, ensuring their validity. Thus, we obtain chromosomes that are homogeneous in their fitness values; the building-blocks (sub-strings) they consist of are diverse enough and f is considerably high. The above holds true even when the total number of updates is large enough and SRA fails to provide good solutions, since the random assignment of 1s in half of the population provides a good "starting point" for GRA. Thus, initialization requires $O(N_p M^2 N + N_p MN^2)$ time.

Selection mechanism: This operation consists 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* as proposed in Holland's SGA (Simple Genetic Algorithm) [17]. This scheme allocates to the i th chromosome, f_i/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 $2\pi f_i/f$ to the i th chromosome and creating 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 in SGA are N_p exactly, i.e., the outcomes of crossover and mutation. Instead of following this approach that can lead to large sampling errors, we selected the *stochastic remainder technique* [14] 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, while 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 [21]. Under this strategy from the initial population of μ size, two more subpopulations are created of total size λ , one from the crossover and the other from the mutation operator. The chromosomes of all these three populations $(\mu + \lambda)$ compete for the μ slots of the next generation (N_p in our case). Finally, we implemented the *elitistic* approach, with which the best chromosome found until one generation before, replaces the worst chromosome of the population. In order to prevent premature convergence, we allow the elite chromosome to be copied back once every 5 generations.

Crossover mechanism: We selected a two-point crossover mechanism to include in GRA. After the pairing of chromosomes two crossover points 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 as to which parts to juxtapose is random. The whole operation is performed with probability μ_c , known as the *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 and we restore their validity by exchanging the portion of the gene that was not previously crossed. The

rationale behind crossover operation, is that after the exchange of genetic materials, it is very likely that the two newly generated chromosomes will possess the good characteristics of both their parents (building-block hypothesis [14][17]).

Mutation mechanism: Mutation is an operation aiming at restoring lost genetic material and is performed in the GRA by simply flipping every bit with a certain probability μ_m , called the *mutation rate*. Mutating a bit can result in violating either the storage, or the primary site constraint. To counter this, we check if either of the two constraints is violated and in such case flip the mutated bit again.

Control Parameters: Large values of μ_c and μ_m force a GA to explore the solution space, while low values favour exploitation. Optimal tuning of these values requires extensive experiments [14]. Typical values of these parameters, as stated in [15] are: $N_p = \{30, 100\}$, $\mu_c = \{0.9, 0.6\}$, $\mu_m = \{0.01, 0.001\}$. Obviously, even with the best decisions on the above parameters, optimal solutions can not be guaranteed due to the algorithm's probabilistic nature. Unless otherwise stated, after considering a series of experimental results GRA's parameters were fixed to: $N_p = 50$, $N_g = 80$, $\mu_m = 0.01$, $\mu_c = 0.9$.

Complexity: Selection is clearly the "hardest" operation because it involves computing D ($O(M^2N)$ time). Thus, GRA runs in $O(N_g N_p M^2 N + \text{initialization})$ resulting in a total time of $O(N_g N_p M^2 N + N_p M N^2)$.

5 Adaptive Genetic Replication Algorithm

So far, we assumed the existence of a monitor site in the network which collects statistics for the objects and defines their replication schemes by using one of the above static methods. Each site sends during night hours the previous day's locally observed R/W patterns to the monitor. After accumulating all the patterns, the monitor site defines new replication schemes using preferably the GRA algorithm. The newly defined schemes are realized during night hours through object migration and deallocation and by the next day the whole network is already tuned accordingly.

The above static methods, however, are not able to deal with the situation when the read-write patterns exhibited during daytime, differ largely from the night time estimations. To cope with the problem, statistics collection should be done every few minutes and the monitor site must be able to quickly decide about new replication schemes. We have already presented SRA and GRA for redistribution of the objects. While the fast execution time of SRA is desirable in a dynamic environment, it can result in low quality schemes which are sometimes worse than the already existing object allocation. On the other hand, the execution time of GRA is unacceptable for the dynamic environment. To overcome this problem, we design another genetic algorithm referred to as AGRA (Adaptive Genetic Replication Algorithm), which rests as a compromise between the two extremes.

Each time the R/W pattern of an object O_k changes above a threshold value either in favour of reads, or updates, AGRA takes as input the new pattern, computes a set of replication schemes R_k for it, which are then transcribed to the initial GRA solutions. The modified population is then inserted to a micro-GRA and evolved for few generations in order to determine an even better solution. R_k s are defined by measuring the NTC occurred from the O_k , without taking the Knapsack component of the DRP into account, i.e., neglecting the storage capacity constraint. Any violations are repaired during the transcription phase. A detailed description of the algorithm follows.

Each chromosome in the population of AGRA is a bitstring of length M . Let O_k be the object for which the algorithm is run. A 1 value in the i th bit of it denotes that $S^{(t)}$ holds a replica of O_k . Let A_p represent the population size and A_g the number of generations it evolves. The initialization of the first generation is performed by randomly generating half of the population while the rest is obtained from the solutions previously found by GRA, making sure that AGRA always copies the current replication scheme of O_k , being incorporated to the highest fitness solution of GRA. The three operations of GA (Selection, Crossover, Mutation) come afterwards to define the population of the next generation.

Let V_k denote the NTC occurred due to reads and updates of the object O_k . V_k can be computed by omitting the summation for all objects $1..N$ in D 's computation (Eq.4). The fitness value we used resembles to the one of GRA and is given by:

$$f_A = \frac{V_{prime} - V_k}{V_{prime}}$$

where V_{prime} stands for the NTC occurred when the only replicator of O_k in the network is its primary site. Following GRA design, in the rare case when a chromosome has $f_A < 0$ we set the chromosome to $f_A = 0$ by setting to 1 only the bit corresponding to SP_k .

The sampling space of AGRA is regular as opposed to GRA where we used enlarged sampling space. It contains all the offspring and some part of the parents (those not subjected to crossover and mutation). Again, stochastic remainder selection 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. Indeed, the selection mechanism used in GRA can result in evaluating three times more chromosomes in the worst case and the improvement in performance if incorporated in AGRA would be marginal, since the algorithm needs to solve an unconstrained problem and operates over chromosomes of small length. From this point of view and by keeping A_p and A_g small (10, 50), AGRA is essentially a micro-GA. We also follow the elitistic approach as in GRA. Single point crossover is used with equal probabilities of crossing the left and the right part of the chromosomes. Mutation again means simply flipping a bit. Constant crossover and mutation rates are used with values of 80% and 1%, respectively.

After A_g generations, AGRA terminates having converged largely to high fitness solutions distributing O_k to a degree that minimizes the NTC of R/W requests. The best R_k found by AGRA is transcribed to half of the initial population of GRA, including the corresponding elite chromosome (current network replica distribution), while the remaining R_k s are randomly transcribed 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 can follow a greedy method and calculate the negative impact each possible one object deallocation has in D value. By deallocating the object which corresponds to the least degradation of D and performing the above steps until the constraint is met, we can have a good quality transcription method. The key drawback though of this method is the high complexity of computing the D value, which can be $O(M^2N)$ in the worst case. Such a computational cost, is clearly unacceptable for a dynamic method.

Instead of deciding which object to deallocate through accurate calculation of its impact to the NTC, we can use a

rapid estimation based on O_k 's degree of replication. Let $S^{(i)}$ be a site where we have allocated more objects than it can accommodate. For each object currently allocated we define a new replica benefit estimation value $E_k^{(i)}$ as:

$$E_k^{(i)} = \frac{(TotalReads + localUpd - TotalUpd) + \frac{localReads \cdot SiteCapacity}{ObjectSize}}{localProportionalLinkWeights \cdot ReplicaDegree}$$

or using the notation of Table 1:

$$E_k^{(i)} = \frac{\left(\sum_{x=1}^M r_k^{(x)} + w_k^{(i)} - \sum_{x=1}^M w_k^{(x)} \right) + \frac{r_k^{(i)} \cdot s^{(i)}}{O_k}}{\sum_{x=1}^M C(i, x)} \cdot \frac{\sum_{x=1}^M X_{xk}}{\left(\sum_{l=1}^M \sum_{x=1}^M C(l, x) \right) / M} \quad (6)$$

The rationale behind the above formula, is that in order to estimate how beneficial a replica is (from NTC reduction standpoint), we must take into consideration both global properties of the object and local characteristics. The global properties consist of whether the object is read demanding or not and how many of its replicas exist in the network. Naturally, an object having high update ratio, but being widely distributed will have more chances of being selected for deallocation. A special weight to the local read requests needs to be given. We do this, in conjunction with the storage capacity of the site and the size of the object. Large objects with poor local read demand are preferred for deallocation, since this will result in freeing up more space for future allocation. Finally, we include an estimation of how good the site acts as the potential closest neighbour of other sites, by calculating its proportional link weights. The computational complexity of the estimation is $O(M)$ for a single object.

Having transcribed the R_k 's 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. In the following section we use and evaluate both policies.

6 Experimental Results

Here, we present the results of our experiments carried out on a 200Mhz Ultra Sparc 2 machine. Five different experiments were conducted. The purpose of the first two is to explore how an increase in the number of sites and objects can affect the performance of the static algorithms. In the next two, we study the effects of the update ratio and the capacity of sites. 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 set we evaluate the performance of AGRA both as a stand-alone algorithm and in combination with the micro-GRA.

6.1 Workload

We generate the network structures in the following manner. First, we define the total number of sites and objects in the network. Among every pair of sites $S^{(i)}$, $S^{(j)}$ there exists a bi-directional link connecting them, with its cost $C(i, j)$ taking values from a uniform distribution 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 [16]. For each object we randomly choose a site to hold the primary copy for it, no other replicas are created.

Next, we generate the number of reads for all (site, object) pairs by using a uniform distribution from 1 to 40. Having defined the read patterns, we use them to calculate the total number of reads for each object, and subsequently, the total number of its updates with respect to U%. The final total update value is defined using a uniform distribution from $TotalUpdates/2$ to $3 \cdot TotalUpdates/2$. We randomly allocate these updates to the sites of the network. In this way we instill enough diversity in our system.

We generate the size of each object using a uniform distribution with mean value 35. In all the experiments, the capacity of sites (C%) is proportional to the total size of objects. The capacity of a site is also generated using a uniform distribution from $C\% \cdot TotalObjectSizes/2$ to $3C\% \cdot TotalObjectSizes/2$. This ensures the creation of sites with diverse enough storing capabilities.

In the fifth experiment we alter the generated R/W patterns to test AGRA. This is performed by adding all the new read requests one by one to randomly chosen sites. For the updates, we follow the same policy for half of them. For the other half, we assign the requests to the sites using normal distribution with variance equal to one fifth of the total number of sites and mean value randomly selected among the number of sites. This is to simulate the scenario when some objects are frequently updated from a specific cluster of nodes and not by all the sites. 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.

6.2 Performance of SRA and GRA

First, we assess the performance of SRA and GRA by varying the number of sites and objects. We fixed C=15% and used three different update ratios, 2%, 5% and 10%. Fig. 1(a)-(d) summarize the results. In the first two figures we study the behavior of the static algorithms when the number of sites increases, by setting the number of objects to 150, while in Fig. 1(c) and (d) we fix the number of sites to 100 and vary the objects from 100 to 1,000. We should note here that experiments with 300 and 400 objects in the first case and 50 and 150 sites in the second, showed similar results concerning the trends in the plots but are not included here due to space limitations. The first observation is that GRA outperforms SRA in terms of solution quality. Only for small network sizes and small update ratio (2%) are the quality of solutions obtained by SRA, comparable to the ones of GRA. Moreover, the savings obtained from the genetic algorithm not only are higher, but tend to diminish less rapidly than the ones yielded by the greedy method. Indeed, Fig. 1(a) and (c) show that the percentage of NTC saving for GRA is only marginally affected by the increase in either the number of sites, or the objects and remain almost constant.

Results in Fig. 1(a) 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, so as to achieve almost constant savings. Indeed, Fig. 1(b) shows an almost linear increase to the number of replicas GRA creates, while the ones from SRA remain constant. The above trend is more obvious when the updates are 2% and 5% which, in terms, means that the additional storage capacity introduced through the increase of sites is explored by GRA when the update ratio is such, that makes replication beneficial. When the update ratio is high, the additional capacity added is only marginally used

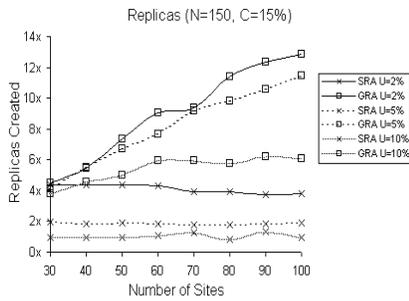


Figure 1(a): Savings in network cost versus the number of sites.

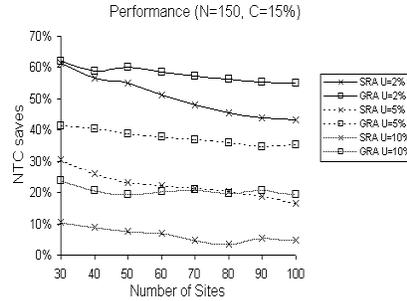


Figure 1(b): The number of replicas generated versus the number of sites.

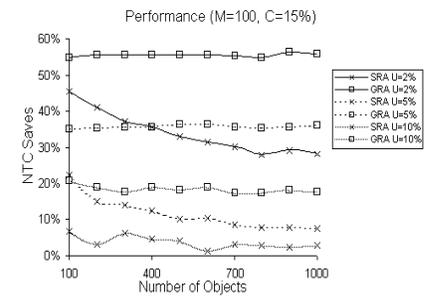


Figure 1(c): Savings in network cost versus the number of objects.

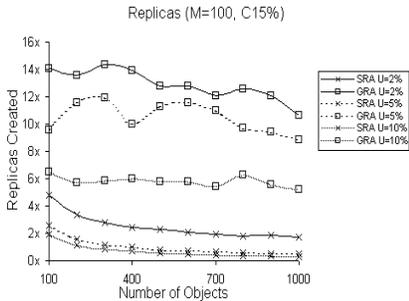


Figure 1(d): The number of replicas generated versus the number of objects.

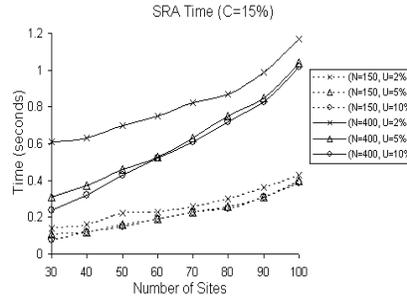


Figure 2(a): Execution time of SRA versus the number of sites.

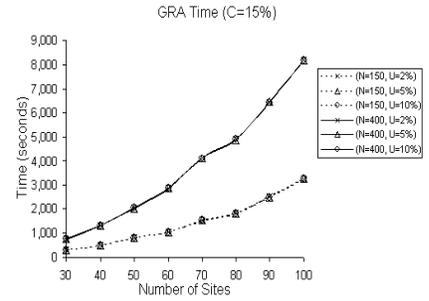


Figure 2(b): Execution time of GRA versus the number of sites.

after a certain point (60 sites in Fig.1(b)), due to the fact that further replicating an object in the network would yield negative results as the updates are high. On the contrary, SRA retains a constant number of replicas, thus, not exploring the additional storage space added.

In the experiments of Fig. 1(c), the capacity of sites is fixed proportionally to the total size of objects, so increasing the number of objects, has no effect to the degree of replication an object can achieve and the benefits it can yield (only depends on the updates). Here also, GRA explores the solution space with the same efficiency as the number of objects grows, while SRA fails to do so.

Without new storage space introduced, GRA manages to retain almost the same degree of replication when $U=10\%$ (Fig.1(d)), resulting in keeping a constant performance. For the smaller update ratios, the degree of replication is not stable but ranges without though affecting the performance. This is due to the non-deterministic nature of GRA. SRA creates far less replicas than GRA (3 times less in the $U=2\%$ case) and the small decrease in the degree of replication experienced, has large impact to the quality of solution.

Unfortunately, the better performance of GRA and the large traffic savings are achieved in expense of large execution time. As shown in Fig. 2(b) GRA's running time is 3-4 orders of magnitude higher compared to SRA and rises in an almost quadratic way to the number of sites, fact that is due to the complexity of the algorithm, see Sec 4. The trend in SRA's execution time in Fig. 2(a), is also quadratic.

The performance of both algorithms decreases exponentially to the update ratio (Fig. 3(a)), with GRA achieving once more better performance than SRA. An increase in storage capacities means that a larger number of objects can be replicated. Since objects are not equally read intensive, increasing the storage capacity has great impact at the beginning, but has little effect after a certain point where the most beneficial ones are already replicated. This is observed in Fig. 3(b) which shows the performance of the

algorithms. The constant NTC savings of SRA in the figure, is due to the fact that with update ratio of 5%, the point where further replicating objects is inefficient, is reached very quickly. Indeed, in further experiments we made with $U = 0.5\%, 1\%$ and 2% , the trends in SRA's performance were similar to the ones of GRA. It is noteworthy that augmenting the capacity from 10% to 30%, resulted in 5 times more replicas being created by GRA, while the relevant performance gains as shown in Fig. 3(b) were less than 1%.

Summarizing the above, GRA achieves more traffic savings (in many cases even 70%), than the greedy method and responds better to changes in the network size, the update ratios or the sites' capacities. On the other hand the greedy method apart from achieving good quality of solutions for small network sizes and update ratios, runs in about 4 orders of magnitude less time.

6.3 Performance of AGRA

Now, we illustrate the performance gains by using the AGRA algorithm. Our test case is a network of $M=50$, $N=200$, $U=5\%$, and $C=15\%$. 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 had 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 200 total objects 48 experienced an increase by 600% in their reads, while 12 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

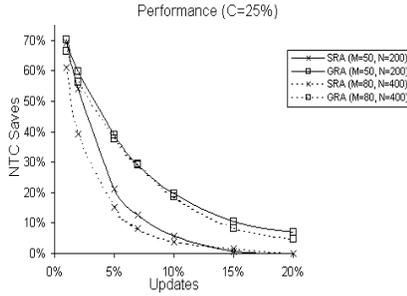


Figure 3(a): Savings in network cost versus the update ratio.

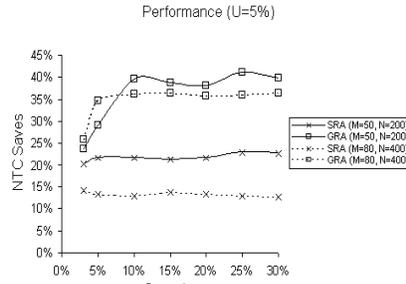


Figure 3(b): Savings in network cost versus the capacity of sites.

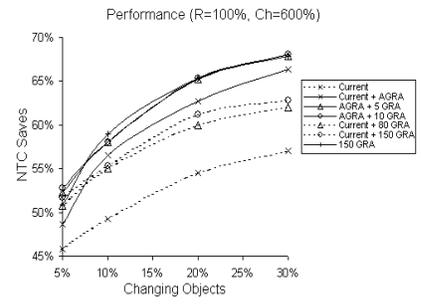


Figure 4(a): Savings in network cost versus the number of objects having their reads increased.

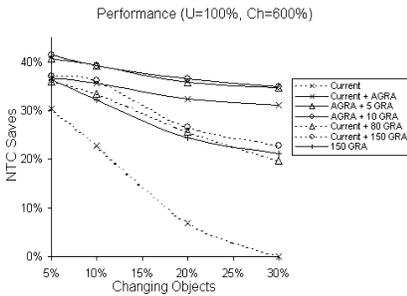


Figure 4(b): Savings in network cost versus the number of objects having their updates increased.

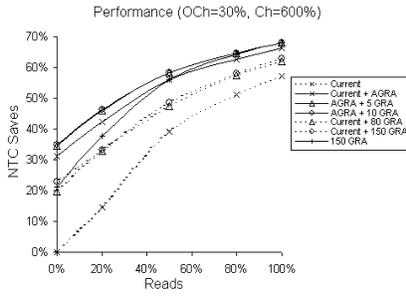


Figure 4(c): Savings in network cost versus the kind of pattern change.

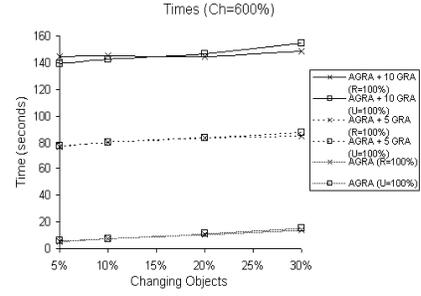


Figure 4(d): Execution time of AGRA versions.

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 executed AGRA with 5 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) and 150 (Current + 150 GRA legend label) generations. Finally, we run 150 generations of GRA (150 GRA legend label), not with the current scheme, but with a population generated from scratch.

The performance of all policies when only reads are increased seems to converge to different upper bounds (Fig. 4(a)). This can be attributed to the fact that an increase in the reads lead all GA policies to 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. 4(b)) due to the fact that increasing the updates of a certain percentage of objects, means only 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. Another thing we should notice is that the initial static solution can be totally outdated and inefficient when updates are significantly increased. For example in Fig.4(b) when the updates are increased by 600% for 20% of the objects, the traffic savings of the previously found replication scheme drops to less than 10%, while AGRA variations achieve traffic savings close to 38%. This fact is supportive for a dynamic method to adapt the replication scheme of the network.

The performance difference between the combinations

of AGRA and the static policies as shown in Fig. 4(a)-(b), is around 10% when large portion of the objects change their patterns, with the (AGRA + 10 GRA) and (AGRA + 5 GRA) achieving comparable savings. Fig. 4(c) shows how the traffic savings increase as the changes in patterns shift from 100% updates to 100% reads. Among all the static methods, running GRA with a random initial population (150 GRA) is the best choice when reads are increased, while the (Current + 80/150 GRA) policies are better when updates increase. These results suggest the usefulness of AGRA and its transcription/estimation method in dynamic environments. As we can observe, 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 1% from (150 GRA) in the case where only reads increase. The (150 GRA) policy though, has prohibitively high execution time around 1920 seconds, while AGRA both as stand-alone and in combination with the mini-GRA is 1.5 to 2 orders of magnitude faster (Fig. 4(d)). The same figure also shows that the increase to the percentage of objects changing patterns has only 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 far better especially when combined with the mini-GRA, since its adaptive/transcription method 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 5 generations of mini-GRA. 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.

7 Related Work

The data replication problem as presented in Section 2 is an extension of the file allocation problem (see [11] for a survey). Chu [10] studied the file allocation problem with respect to multiple files in a multiprocessor system. Casey [8] extended this work by distinguishing between updates and queries on files. Eswaran [12] proved that Casey's formulation was NP complete. Apers [3] considered the allocation of data in distributed databases, so as to minimize the total data transfer cost, while Kwok *et al.* [18] proposed several algorithms to solve the data allocation problem (without replication) in distributed multimedia databases. Our model is unique as we consider the capacity constraint and the issue of multiple copies at the same time, reflecting a pragmatic scenario in today's distributed information environments such as the Internet. Moreover, previous works assume a static environment and solve the problem mainly by using integer programming or similar time consuming techniques.

Some recent work [5], [22] addresses dynamic replication of objects in distributed systems when the read-write patterns are not known *a priori*. Although Awerbuch's *et al.* work [5] is significant from a theoretical point of view, the competitive algorithm proposed has little practical applications, since after a write is issued all replicas but one are deleted. In [22] Wolfson *et al.* proposed an algorithm which leads to optimal single file replication in case of a tree network. However, the performance of the scheme for cases other than the tree networks is not clear. In [20] the authors proposed a protocol for replication over the Internet. In the model used though, no attention was paid to the update cost, thus limiting its scope.

Genetic algorithms have been used to solve various optimization problems including graph partitioning [9] and multiprocessor document allocation [13]. We took 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.

8 Conclusions

In this paper we addressed the data replication problem and developed a cost model, which is applicable to very large distributed systems such as the WWW and distributed databases. 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 GRA constantly outperforms SRA 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 proposed 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 a promising choice in a dynamic environment such as the Internet.

References

- [1] M. Abrams, C. Standridge, G. Abdulla, S. Williams and E. Fox, "Caching Proxies: Limitations and Potentials", *Electron. Proc. 4th World Wide Web Conf'95: The Web Revolution*, Boston MA, Dec. 11-14, 1995.
- [2] T. Anderson, Y. Breitbart, H.F. Korth and A. Wool, "Replication, Consistency and Practicality: Are These Mutually Exclusive?", *ACM SIGMOD'98*, Seattle, June 1998.
- [3] P.M.G. Apers, "Data Allocation in Distributed Database

- Systems," *ACM Trans. Database Systems*, 13(3), Sep. 1988, pp. 263-304.
- [4] M.F. Arlitt and C.L. Williamson, "Internet Web Servers: Workload Characterization and Performance implications", *IEEE/ACM Trans. on Networking*, Vol. 5, No. 5, pp. 631-645, Oct. 1997.
- [5] B. Awerbuch, Y. Bartal and A. Fiat, "Optimally-Competitive Distributed File allocation", *25th Annual ACM STOC*, Victoria, B.C., Canada, 1993, pp. 164-173.
- [6] M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm, "Enhancing the web infrastructure - from caching to replication", *IEEE Internet Computing*, pp. 18-27, Mar-Apr 1997.
- [7] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-based Caching", *IEEE Concurrency: Special Issue on Parallel and Distributed Technology*, Vol.5, pp. 56-67, Jan.-Mar. 1997.
- [8] R.G. Casey, "Allocation of Copies of a File in an Information Network," *Proc. Spring Joint Computer Conf., IFIPS*, 1972, pp. 617-625.
- [9] R. Chandrasekharam, S. Subramanian, and S. Chaudhury, "Genetic Algorithm for Node Partitioning Problem and Applications in VLSI Design," *IEEE Proceedings*, vol. 140, no. 5, Sep. 1993, pp. 255-260.
- [10] W.W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Trans. Computers*, vol. C-18, no. 10, 1969.
- [11] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment problem", *ACM Computing Surveys*, Vol.14(2), June 1982.
- [12] K.P. Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing*, 1974, pp. 304-307.
- [13] O. Frieder, H.T. Siegelmann, "Multiprocessor Document Allocation: A Genetic Algorithm Approach", *IEEE Trans. on Knowledge and Data Eng.*, vol.9, no.4, July/Aug. 1997, pp.640-642.
- [14] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley*, Reading, Mass., 1989.
- [15] J.J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. Systems, Man and Cybernetics*, Vol.SMC-16, No.1, Jan./Feb. 1986, pp. 122-128.
- [16] J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching", *Proc. 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, IEEE Computer Society Press, Los Alamitos, Calif., 1995, pp.51-55.
- [17] J.H. Holland, "Adaptation in Natural and Artificial Systems", *University of Michigan Press*, Ann Arbor, Mich., 1975.
- [18] Y.K. Kwok, K. Karlapalem, I. Ahmad and N.M. Pun, "Design and Evaluation of Data Allocation Algorithms for Distributed Database Systems", *IEEE Journal on Sel. areas in Commun.(Special Issue on Distributed Multimedia Systems)*, Vol. 14, No. 7, pp. 1332-1348, Sept. 1996.
- [19] S. Martello and P. Toth, "Knapsack Problems: Algorithms and Computer Implementations", *John Wiley & Sons-Interscience Series in Discrete Mathematics and Optimization*, 1990.
- [20] M. Rabinovich, I. Rabinovich, R. Rajaraman and A. Aggarwal, "A dynamic object replication and migration protocol for an Internet hosting service." *IEEE Int. Conf. on Distributed Computing Systems*, May 1999.
- [21] H. Scwefel, "Evolution and Optimum Seeking", *John Wiley & Sons*, New York, 1994.
- [22] O. Wolfson, S. Jajodia, Y. Huang, "An Adaptive Data Replication Algorithm", *ACM Transactions on Database Systems*, Vol. 22(4), June 1997, pp. 255-314.