Formal Model and Scheduling Heuristics for the Replica Migration Problem

Nikos Tziritas[1], Thanasis Loukopoulos[1], Petros Lampsas[2], and Spyros Lalis[1]

[1] Dept. of Computer and Communication Engineering, University of Thessaly, Glavani 37, 38221 Volos, Greece
{nitzirit,luke,lalis}@inf.uth.gr
[2] Dept. of Informatics and Computer Technology, Technological Educational Institute (TEI) of Lamia, 3rd km. Old Ntl. Road Athens, 35100 Lamia, Greece
plam@teilam.gr

Abstract. Replication of the most popular objects is often used in distributed data provision systems to reduce access time and improve availability. In fact, a given replica placement scheme may have to be redefined as object popularity changes. Given two replica placement schemes X^{old} and X^{new}, the Replica Migration Problem (RMP) is to compute a schedule of replica transfers and deletions that lead from X^{old} to X^{new} in the shortest time possible. In this paper, we provide a rigorous problem formulation and prove that even for trivial cases RMP is intractable. We also propose a set of heuristics and evaluate them for different scenarios using simulations.

1 Introduction

Replication is crucial to the performance of distributed data provision systems such as web and video server networks. The issue of defining a replication scheme, i.e. which data objects to replicate on which nodes, was studied extensively under the context of the replica placement problem (RPP) [1], [2], [3], [4]. Equally important, however, is to derive an implementation strategy for obtaining the desired replication scheme. The problem can be briefly stated as: *given two replication schemes X^{old} and X^{new}, find a series of object transfers and deletions that lead from X^{old} to X^{new} in the shortest time possible*. We refer to this as the Replica Migration Problem (RMP).

RMP has been tackled in [5] and [6] with focus primarily on disk farms, while [7] focuses on content distribution networks. In all these cases the aim is to optimize the migration time. More recently the problem has been incorporated in task scheduling over the Grid [3] where the aim is to minimize the final makespan of task executions. The problem is also studied in [8] but with the aim to minimize network usage. In [5] and [7] object sizes are assumed to be equal, while in [5] and [6] hosting nodes are assumed to be fully connected. We differ from the above work both in the scope of the problem assumptions (which are more general) and the heuristics we propose.

Our contributions are as follows: (i) we provide a rigorous formulation of RMP as a mixed integer programming problem with some of the constraints being quadratic; (ii) we prove that even for trivial cases RMP-decision is NP-complete given different object sizes; (iii) we propose and experiment with heuristics capturing different problem parameters such as deletions and the creation of auxiliary replicas. To the best of our knowledge this is the first time RMP is stated and tackled in this way, with the aim of minimizing replica migration time under various important parameters such as object size, network bandwidth and storage space.

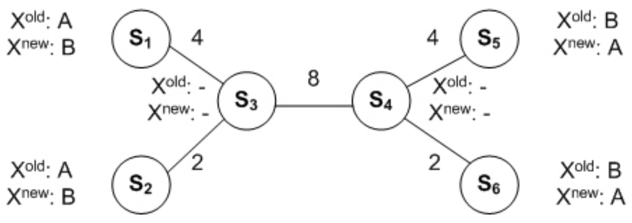
The rest of the paper is organized as follows. Section 2 presents the system model, and Section 3 gives the formal problem statement. Then, Section 4 describes our heuristics, which are evaluated via simulations in Section 5.

2 Problem Description

Consider a distributed system with M servers and N data objects. Let S_i and $s(S_i)$ denote the name and the storage capacity (in abstract data units) of the i^{th} server, $1 \le i \le M$. Also, let O_k and $s(O_k)$ denote the k^{th} data object and its size, $1 \le k \le N$. We say that S_i is a *replicator* of O_k if it holds a replica thereof. Let X be a $M \times N$ *replication matrix* used to encode a replication scheme as follows: $X_{ik}=1$ if S_i is a replicator of O_k, else $X_{ik}=0$. Servers communicate via point-to-point *links*. A link between S_i and S_j, if it exists, is denoted by l_{ij} and has a capacity of c_{ij}, representing the number of data units that can be transferred via the link per (abstract) time unit. Let T_{ikj} denote the transfer of object O_k from source S_i to destination S_j. This involves sending $s(O_k)$ data units along a path from S_i to S_j. The transfer will complete after $s(O_k)/r$ time units, where r is the transfer rate of that path, equal to the available capacity of the bottleneck link. Both c_{ij} and r are integers, denoting multiples of one data unit/time unit. Finally, the deletion of O_k at S_i, denoted by D_{ik}, does not introduce any time penalty.

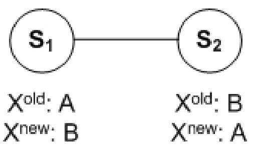
The problem we tackle is to define a series of transfers and deletions so that starting from the current replication scheme X^{old} we reach a new replication scheme X^{new} in the shortest time. We illustrate it through the example of Figure 1, using a network of 6 servers with 2 objects: A and B. In X^{old}, S_1 and S_2 hold object A whereas S_5 and S_6 hold object B. Objects must be swapped in X^{new}. Assuming object size and server capacity of 4, schedule $\{D_{5B}, T_{1A5}, D_{2A}, T_{6B2}, D_{6B}, T_{1A6}, D_{1A}, T_{2B1}\}$ implements X^{new} in 6 time units. The start and end time of the transfers are listed in the table of Figure 1. Transfers can occur in parallel, even if their paths overlap, provided that there is enough capacity available. Specifically, in this schedule, T_{1A5} via links l_{13}, l_{34} and l_{45} is performed in parallel to T_{6B2} via links l_{64}, l_{34} and l_{32}, with a rate of 4 and 2, respectively. However, if link l_{34} had a capacity of just 5 instead of 8, at least one of these transfers would have been performed at a lower rate, if done in parallel.

It is important to note that it is not always possible to find a schedule, even for valid problem statements where X^{new} respects the server capacity constraints. For example, if two servers with enough storage capacity to hold just one object must exchange their objects, and there are no other servers that can be used as a source for these objects, a deadlock-lie situation occurs, as shown in Figure 2. The investigation of such cases is out of the scope of this paper. Therefore we extend the problem formulation by assuming that there is one *primary replica* for each object O_k that remains fixed on a designated *primary server* P_k in both X^{old} and X^{new}.



Transfer	*start*	*end*
T_{1A5}	0	1
T_{6B2}	0	2
T_{1A6}	2	4
T_{2B1}	4	6

Fig. 1. An example problem instance



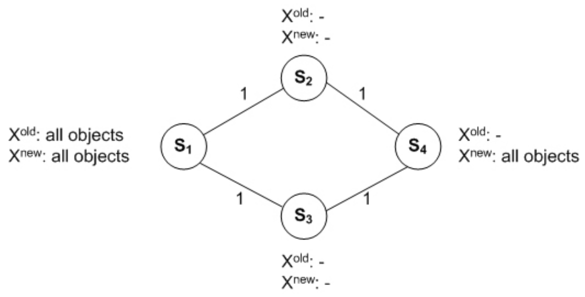


Fig. 2. A deadlock example

Fig. 3. Network construction for 2-proc reduction

Continuing with the example of Figure 1, one may observe that the aforementioned schedule is not optimal. The optimal schedule is $\{T_{1A4}, T_{5B3}, D_{1A}, D_{2A}, D_{5B}, D_{6B}, T_{4A5}, T_{4A6}, T_{3B1}, T_{3B2}, D_{4A}, D_{3B}\}$ with a makespan of 3 time units. This schedule is non-trivial in the sense that it involves *auxiliary* replicas, i.e., replicas that are not required in X^{new} and will be deleted at some point further in the schedule.

In fact RMP-decision is intractable. Due to space limitations we provide a proof sketch by reducing the 2-processor scheduling problem (2-proc) [9], which is defined as follows: *given a set of tasks N and their execution times, assign them to 2 processors so that the total makespan is minimized.* For each 2-proc instance we construct the network of Figure 3, where *N* objects exist and their sizes correspond one to one with the execution times of the *N* tasks. X^{old} consists of primary replicas stored at S_1, while in X^{new} one additional replica per object must be created at S_4. Notice that the two paths connecting S_1 and S_4 play the role of the two processors in 2-proc. As a consequence, there exists a solution to 2-proc if and only if a solution exists for RMP-decision in the problem instance of Figure 3. Thus, RMP-decision is NP-complete.

3 Integer Programming Formulation

We formulate RMP as a mixed integer programming (MIP) problem with some constraints being quadratic. The idea is to consider a schedule of transfers and deletions, similar to the example of Figure 1, and impose the respective validity requirements. Each transfer is modelled using a *transfer-start* and a *transfer-end* event. Deletions correspond to a single *delete* event.

The formulation assumes a known upper bound for the number of events in the optimal schedule, let Z. An additional (dummy) void event is needed at the end of the schedule in order to confirm that the desired replication matrix has been reached after the Zth event. Thus, the total length of the schedule is $Z+1$, its tail comprised of at least one void event (or more if the actual optimal schedule involves fewer than Z events).

For the case where no auxiliary replicas are created, a conservative value for Z is: *2* × *outstanding replicas* + *superfluous replicas* (outstanding are the new replicas that must be created and superfluous are the old replicas that need to be deleted to reach X^{new}). This corresponds to the number of required transfers (2 events per transfer) and deletions (1 event per deletion), which can be trivially determined as a function of X^{old} and X^{new}. In case auxiliary replicas can be created, the upper bound for Z is: *2* × *outstanding replicas* + *superfluous replicas* + *3* × *auxiliary replicas* (for each auxiliary replica, 2 events are needed to create it, and 1 event is needed to delete it). Given that the number of auxiliary replicas for each outstanding replica cannot exceed (M-*2*) (worst case, an auxiliary replica is created on all servers except the primary server and the server where the outstanding replica is to be created): $Z \leq (3M\text{-}4)$**outstanding* + *superfluous*. Note that these bounds do not hold for schedules that may contain deletions and subsequent re-creations of replicas needed in X^{new}.

Table 1 summarizes the variables used to describe the events in the schedule as well as additional problem variables. Unless otherwise stated, the indices in the variables take the following values: $1 \leq k \leq N$, $1 \leq i,j \leq M$, $1 \leq u,v \leq Z$.

Table 1. Problem variables

Variable	Description
ST^{u}_{ikj}	1 iff the uth event is the start of transfer T_{ikj}, 0 otherwise
ET^{uv}_{ikj}	1 iff the uth event is the end of transfer T_{ikj} whose start is the vth event in the schedule ($v<u$), 0 otherwise
D^{u}_{ik}	1 iff the uth event is the deletion of O_k at S_i, 0 otherwise
V^{u}	1 iff the uth event is void, 0 otherwise
X^{u}_{ik}	1 iff S_i has a replica of O_k before the uth event occurs, 0 otherwise
p^{u}_{ij}	1 iff the uth event is a transfer start/end event and link l_{ij} is part of the corresponding transfer path, 0 otherwise
r^{u}	the transfer rate for the uth event (integer$\neq$0)
t^{u}	the clock time when the uth event takes place (real$\geq$0)

The Replica Migration Problem can then be stated as: *minimize* t^{Z+1} *subject to constraints (1)-(11).* Constraints (1)-(6) relate event types among themselves and with the current replication matrix, (7)-(9) concern path reservations for transfers, while (10)-(11) tackle bandwidth assignment and time calculation.

$$\sum_{\forall i}\sum_{\forall k}\sum_{\forall j}\sum_{\forall v} ET_{ikj}^{uv} + \sum_{\forall i}\sum_{\forall k}\sum_{\forall j} ST_{ikj}^{u} + \sum_{\forall i}\sum_{\forall k} D_{ik}^{u} + V^{u} = 1 \ \forall u\ , \ \ V^{Z+1} = 1\ . \tag{1}$$

$$X_{ik}^{1} = X_{ik}^{old} \ \forall i,k\ , \ \ X_{ik}^{Z+1} = X_{ik}^{new} \ \forall i,k \ , \ X_{P_k k}^{u} = 1 \ \forall k\ , \ \sum_{\forall k} X_{ik}^{u} s(O_k) \le s(S_i) \ \forall i,u\ . \tag{2}$$

$$X_{jk}^{u+1} = X_{jk}^{u} + \sum_{\forall i}\sum_{\forall v} ET_{ikj}^{uv} - D_{jk}^{u} \ \forall j,k,u\ . \tag{3}$$

$$ST_{ikj}^{u} \le X_{ik}^{u} \ \forall i,k,j,u \ , \ ST_{ikj}^{u} \le 1 - X_{jk}^{u} \ \ \forall i,k,j,u\ , \ D_{ik}^{u} \le X_{ik}^{u} \ \forall i,k,u\ . \tag{4}$$

$$ST_{ikj}^{u} = \sum_{v>u} ET_{ikj}^{vu} \ \forall i,k,j,u \ , \ \sum_{\forall i}\sum_{\forall k}\sum_{\forall j}\sum_{\forall u} ST_{ikj}^{u} = \sum_{\forall i}\sum_{\forall k}\sum_{\forall j}\sum_{\forall u}\sum_{\forall v} ET_{ikj}^{uv}\ . \tag{5}$$

$$\sum_{\forall j}\sum_{\forall v<u} ST_{ikj}^{v} - \sum_{\forall j}\sum_{\forall v<u}\sum_{\forall x | v<x<u} ET_{ikj}^{xv} \le 1 - D_{ik}^{u} \ \forall i,k,u\ . \tag{6}$$

$$p_{si}^{u} = \sum_{\forall k}\sum_{\forall j} ST_{ikj}^{u} + \sum_{\forall k}\sum_{\forall j}\sum_{\forall v<u} ET_{ikj}^{uv} \ \forall i,u\ . \tag{7}$$

$$p_{jd}^{u} = \sum_{\forall k}\sum_{\forall i} ST_{ikj}^{u} + \sum_{\forall k}\sum_{\forall i}\sum_{\forall v<u} ET_{ikj}^{uv} \ \ \forall j,u\ , \ \sum_{\forall i} p_{ij}^{u} = \sum_{\forall x} p_{jx}^{u} \ \forall j,u\ . \tag{8}$$

$$p_{xy}^{u} - p_{xy}^{v} \le 1 - \sum_{\forall i}\sum_{\forall k}\sum_{\forall j} ET_{ikj}^{uv} \ \forall x,y,u,v\ , \ p_{xy}^{u} \le 1 - \sum_{\forall i}\sum_{\forall k} D_{ik}^{u} - V^{u} \ \forall x,y,u \ . \tag{9}$$

$$ET_{ikj}^{uv}(r^{v} + r^{u}) = 0 \ \forall i,k,j,u,v\ , \ \ r^{u} p_{ij}^{u} + \sum_{v<u} r^{v} p_{ij}^{v} + \sum_{v<u} r^{v} p_{ji}^{v} \le c_{ij} \ \ \forall i,j,u\ . \tag{10}$$

$$t^{u} \le t^{u+1} \ \ \forall u\ , \ ET_{ikj}^{uv}(t^{u} - t^{v} - \frac{s(O_k)}{r^{v}}) = 0 \ \forall i,k,j,u,v\ . \tag{11}$$

Constraints (1) state that each event is either a transfer start, transfer end, deletion or void, and that the last event is void. (2) requires that the replication scheme starts with X^{old} and reaches X^{new} without deleting primary replicas or violating server capacities. (3) captures the bit flips in the replication matrix, i.e. if the uth event is a transfer end, then the resulting matrix should have a value of 1 at the corresponding cell; if it is a deletion it should have a 0, while in all other cases the matrix cell should remain unchanged. Since the variables of (3) are binary, the uth event cannot be a transfer-end if $X_{jk}^{u} = 1$, and similarly it cannot be a deletion if $X_{jk}^{u} = 0$. (4) states that in order for a transfer to start, the source server must have a copy of the object and the destination server must not, while for deletions the server must have the replica to be deleted. (5) states that each start event must have a corresponding end event that occurs later in the schedule, and that the number of end events must be equal to the number of start events, to avoid having orphan end events. (6) ensures that the source of a transfer cannot be deleted before that transfer ends.

Next come constraints related to path reservation (7)-(9). Whenever a transfer start/end event occurs, a path from the source to the destination must be reserved (7)-(8). To do this, we assume that all transfers start from a dummy source (7) and end to a dummy destination (8). These dummy servers are connected through direct links of unlimited capacity to all other servers. To guarantee that a path is not interrupted, (8) requires that if a server (other than the dummies) has an incoming link belonging to the path, it should also have an outgoing. (9) demands that corresponding start and end events have the same path, while deletion and void events do not have a path.

Constraints (10) capture bandwidth management aspects. The first part states that the sum of the rates of corresponding transfer start and end events equals zero, ensuring that bandwidth is properly freed when a transfer completes. The second part requests that the aggregate rates of current and past events do not exceed link capacity. The final set of constraints (11) keeps track of clock time. The first part states that events must be properly ordered in time, while the second part calculates the time of an end event as the sum of the start time and the transfer duration. Note that the rate of a start event must be positive (and thus the rate of an end event must be negative), else the end event would occur prior to the start event in the schedule.

We have implemented this MIP problem in LINDO [10], a commercial optimizer. Unfortunately, we were able to obtain solutions, within acceptable time, only for very small problem sets (around 5 objects and servers).

4 Scheduling Heuristics

Given the computationally intensive nature of RMP, we have designed a set of heuristics that can be used to produce solutions for this problem. These heuristics follow the same generic algorithmic template, described in the following pseudocode:

```
(1)  while (outstanding replicas exist)
(2)    for each outstanding replica
(3)      for each source for this outstanding replica
(4)        select a transfer using <selection criterion>
(5)      end for
(6)      apply cut-off rule
(7)      select source using <selection criterion>
(8)    end for
(9)    choose transfer/s with the earliest completion time
(10)   choose transfer/s that require no deletions, if any
(11)   choose transfer/s with the smallest hop count
(12)   select a single transfer by breaking ties randomly
(13)   if (free storage at destination less-than object size)
(14)     perform deletion(s) according to <deletion rule>
(15)   end if
(16)   schedule the selected transfer
(17) end while
(18) delete remaining superfluous replicas
```

In a first phase, for each outstanding replica and possible source for it, one transfer path is chosen subject to a *selection criterion* (lines 2-4). As a result, a set of tuples of the form <outstanding replica; source; path> is produced. Then, a cut-off rule is applied to eliminate some candidates (line 6). Specifically, all transfers are discarded that have paths with an intermediate node that is either a replicator or a host for an outstanding replica of the object to be transferred. The rationale is to favor transfers that do not occupy many links. Then, the same *selection criterion* of line 4 is applied (again) to choose one <source; path> candidate per outstanding replica (line 7).

In a second phase, out of the set of candidate transfers, one per outstanding replica, the ones with the earliest completion time are selected (line 9). Among the remaining candidates, the ones that require no deletions are chosen, and from those, preference is given to transfers having the shortest path hop-wise (line 11). Ties are randomly broken. Before scheduling the chosen transfer, in a third phase, the remaining storage capacity at the destination is checked (lines 12-13). If needed, deletions of superfluous replicas are performed at the server, according to a *deletion rule*.

These three phases are repeated until there are no more outstanding replicas, i.e. all transfers have been scheduled. Finally, the remaining superfluous replicas are deleted (line 18), at no extra cost. Given this template, the additional aspects of our heuristics are introduced below.

Per Object variants *(O-)*: This family of heuristics follows the generic template, but on a per-object basis (imagine an extra outer loop, iterating over all objects added in the generic algorithmic template). This considerably shrinks the search space for selecting a transfer, reducing the running time, at the risk of producing inferior results, compared to the default "across-objects" versions. Still, there is an (indirect) advantage: once the outstanding replicas of the selected object are created, all its superfluous replicas can be safely deleted, knowing that they will never be used as sources for future transfers. Objects are considered in descending size of the respective transfer volume (*outstanding replicas* × *size*).

Selection criterion options: The criterion for selecting transfers (in lines 4 and 7) is either "earliest start time" (EST) or "earliest completion time" (ECT), computed based on the transfers that have been scheduled and the available link bandwidth.

Deletion rule options: The default option is to choose randomly among the superfluous replicas. The second option (denoted with *-d*) is to delete them in ascending order of their *benefit/size* ratio. Intuitively, the benefit of a replica indicates its importance as a potential source for future transfers. For each outstanding replica for which the replica in question is the best source according to ECT rule (given an unloaded network), the second-best source is found, and the time difference is computed between completing the transfer from the best and from the second-best source, respectively. The aggregated time differences, give the benefit. If a replica is not the best source for any outstanding replica, its benefit is zero.

Auxiliary Replica Creation operator *(-ARC)*: This can be applied once the transfer to be performed is chosen (right after line 12). It checks the inner nodes of the respective path, assessing whether any of them could be used to create a *useful* superfluous replica, as follows. For each additional outstanding replica, the ECT transfer source and path is defined (given an unloaded network). If a node is found in the path of at least two such transfers, it is considered as a candidate, provided it has sufficient storage. If such nodes exist, the one closest (in terms of hops) to the source of the transfer to be performed is selected (the original transfer is dropped), a transfer is scheduled to create an auxiliary replica on it, and the algorithm continues with a new iteration.

Our heuristics are built as different combinations of the above options. Henceforth, we refer to heuristics via acronyms, e.g., EST refers to the heuristic that uses the default template, EST as the selection criterion and the default deletion rule, while O-ECT-ARC-d refers to the per-object heuristic that uses ECT as the selection criterion, the benefit-driven deletion rule and the auxiliary replica creation operator.

5 Experiments

In a first set of experiments, 5 networks of 25 servers with connectivity 1 (tree network) and 5 networks with connectivity 3 were generated using BRITE [11], under the Barabasi-Albert connectivity model [12]. Link capacities and object sizes were set equal to 1 and the number of objects was set to 100.

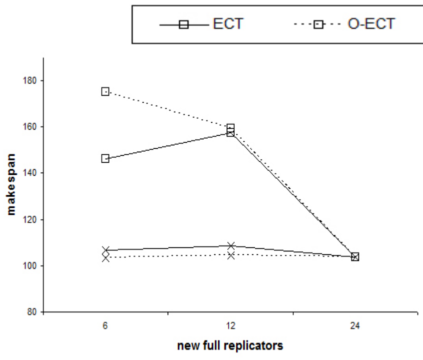


Fig. 4. Replica creations (connectivity 1)

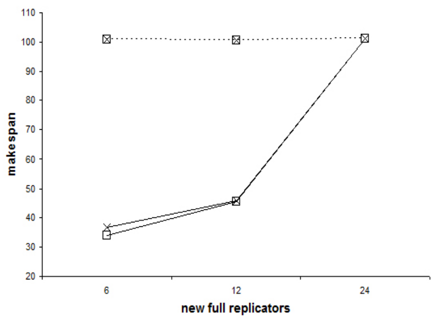


Fig. 5. Replica creations (connectivity 3)

Figures 4 and 5 depict the average makespan for the networks with connectivity 1 and 3, respectively, when starting from a single primary server that holds a replica for each object, and creating additional replicas of all objects to 6, 12 and 24 servers (corresponding to 25%, 50% and respectively 100% of the initially empty servers).

As it can be seen in Figure 4, for the tree networks the makespan of ECT and O-ECT drops as more replicas need to be created. This apparently counterintuitive result hints to the merits of creating auxiliary replicas when a limited number of disjoint paths exist, as illustrated by the superior performance of their ARC counterparts. It is also interesting to observe that these variants result in an almost constant makespan, independently of the number of replicas to be created. This indicates that once auxiliary replicas are created at key locations, the creation of even more outstanding replicas can be achieved at a small additional cost. The non-ARC variants have almost equal performance to the ARC variants when replicas must be created on all servers. Indeed, in this particular case, the non-ARC variants unavoidably create (outstanding) replicas on every server, including the locations that are selected by ARC to create the auxiliary replicas for the cases that require a smaller amount of replication.

However, the benefits of the ARC variants diminish for larger connectivity, as illustrated in Figure 5. This is due to the fact that more outstanding replicas can be created with single-hop transfers, reducing the importance of creating auxiliary replicas. It can also be seen that ECT clearly outperforms O-ECT. This is explained by

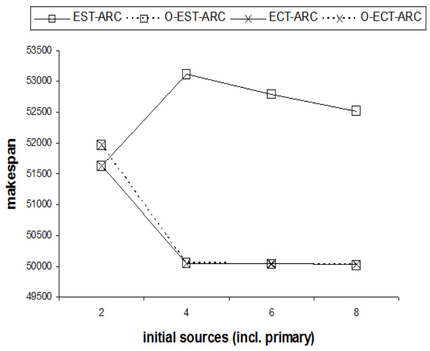


Fig. 6. Replica creations (GÉANT2)

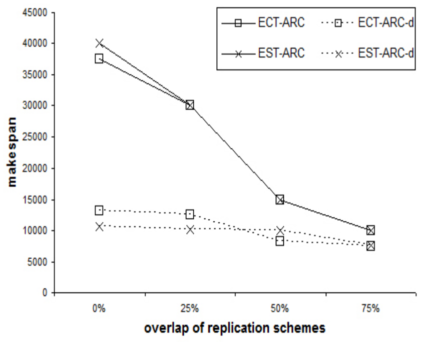


Fig. 7. Replica creations+deletions (GÉANT2)

noticing that the makespan of O-ECT is around 100, corresponding to 1 time unit per object, which means that only one object is replicated at a time. On the contrary, ECT aggregates transfers of different objects in each time slot, taking full advantage of alternative paths that exist in the network. This leads to the creation of new replicas that can be used as sources in subsequent time slots, thus maximizing link utilization.

The next set of experiments was performed using a real-world topology, namely the European research and education network GÉANT2 [13]. The relative link capacities were derived from GÉANT2, while the size of objects was set so that a transfer over the fastest link requires 10 time units. Again, the number of objects was 100.

Figure 6 shows the performance of EST-ARC, ECT-ARC and their per-object variants, when starting from a given number of randomly selected servers (in addition to the primary server) that hold replicas of all objects, and creating additional replicas of all objects on 50% of the remaining (initially empty) servers. Results show that ECT-ARC and O-ECT-ARC achieve comparable performance, closely followed by O-EST-ARC. The trends for the first three algorithms are decreasing as the number of initial sources increases, which can be attributed to the increased link utilization when more initial sources are available. EST-ARC is clearly inferior. This is due to the fact that as the potential sources for an object transfer increase, the earliest starting transfer is also more likely to cross a path with relatively limited capacity, thereby increasing the overall makespan.

Last, we assess the performance of the benefit-based deletion criterion. For this purpose we set the capacity of each server to allow the hosting of 10 objects, initially place on each server a replica of 10 randomly selected objects, and vary the replica overlap between X^{old} and X^{new}: an overlap of 0% means that all replicas in X^{new} are stored in different locations compared to X^{old} (except for the primary replicas), while an overlap of 75% means that only 1 out of 4 replicas needs to be relocated in X^{new}. The results are depicted in Figure 7. As it can be seen, benefit-based deletion clearly outperforms random deletion, for both EST-ARC and ECT-ARC, when a large number of deletions need to be performed, i.e., for small replica overlaps. The performance gap shrinks as the number of required deletions drops.

As a conclusion (also based on additional experiments not shown here due to lack of space), the heuristics which create auxiliary replicas and employ the benefit-based deletion rule achieve better overall results. The per-object variants generally perform worse than their counterparts, but have a significantly reduced execution time (5 to 100 times faster, depending on the experiment).

References

1. Khan, S., Ahmad, I.: A Semi-Distributed Axiomatic Game Theoretical Mechanism for Replicating Data Objects in Large Distributed Computing Systems. In: Proc. 21st Int. Parallel and Distributed Processing Symp (IPDPS 2007), Long Beach, California (March 2007)
2. Loukopoulos, T., Lampsas, P., Ahmad, I.: Continuous Replica Placement Schemes in Distributed Systems. In: Proc. 19th ACM International Conference on Supercomputing (ACM ICS), Boston, MA (June 2005)
3. Desprez, F., Vernois, A.: Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid. Report RR2005-01, INRIA, France (January 2005)
4. Laoutaris, N., Smaragdakis, G., Bestavros, A., Matta, I., Stavrakakis, I.: Distributed Selfish Replication. In IEEE Trans. on Parallel and Distributed Systems (TPDS) 17(12), 1401–1413 (2006)
5. Hall, J., Hartline, J., Karlin, A., Saia, J., Wilkes, J.: On algorithms for efficient data migration. In: Proc. of the twelfth annual ACM-SIAM Symposium on Discrete algorithms (SODA 2001), Washington D.C., United States, pp. 620–629 (2001)
6. Khuller, S., Kim, Y., Wan, Y.: Algorithms for data migration with cloning. In: Proc. of the twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2004), pp. 448–461 (2004)
7. Killian, C., Vrable, M., Snoeren, A., Vahdat, A., Pasquale, J.: Brief Announcement: The Overlay Network Content Distribution Problem. In: Proc. ACM Symp. on Principles of Distributed Computing (PODC), Las Vegas, NV, July 2005, p. 98 (2005)
8. Loukopoulos, T., Tziritas, N., Lampsas, P., Lalis, S.: Implementing Replica Placements: Feasibility and Cost Minimization. In: Proc. 21st Int. Parallel and Distributed Processing Symp (IPDPS 2007), Long Beach, California (March 2007)
9. Garey, J., Johnson, D.: Computers and Intractability. W. H. Freeman and Co., NY (1979)
10. `http://www.lindo.com`
11. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: Boston University Representative Internet Topology Generator (March 2001), `http://cs-pub.bu.edu/brite/index.htm`
12. Barabasi, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science 286, 509–512 (1999)
13. GÉANT2, Backbone Network Topology (January 2008), `http://www.geant2.net/upload/pdf/PUB-07-179_GN2_Topology_Jan_08_final.pdf`