On minimizing memory and computation overheads for binary-tree based data replication

Stavros Souravlas*, Angelo Sifaleras†
Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
Email: *sourstav@uom.gr, †sifalera@uom.gr

***Abstract*—Data replication is used to track the most popular files (i.e., the ones with most requests) and replicate them in selected nodes. In this way, more requests for such popular files can be completed over a period of time and bandwidth consumption is reduced, since these files do not need to be transferred from remote nodes. In this article, we extend our previous work [1] to make it more efficient in terms of memory and total computation cost, so that it becomes more efficient and suitable for larger grids. To reduce the memory costs, we present a centralized strategy which estimates the potential for selected batches of files. The computations required for these estimations are executed in a pipelined way, so their cost is also reduced.**

***Index Terms*—Replication Optimization, Data Grid, Pipeline, Estimation**

I. Introduction

Many applications are moving towards distributed interconnected environments like data grids. As such systems expand, the grid users suffer long delays when they try to access the data. In such environments, data replication is necessary, so that the users can retrieve the requested data from nearby instead of faraway nodes. The idea behind data replication is based on the temporal locality principle [2], [3], which states that some data of a grid is accessed and transferred more frequently, at least over periods of time.

A good number of data replication strategies has been proposed by several researchers [4], [5], [6], [7], [8], [9], [10]. These strategies mainly focus on the problem of selecting the most suitable nodes for storing the replicas. The decision regarding the files to be replicated is based on the number of requests for the file. However, when working only with the number of requests, it is impossible to spot files that have increasing demand, that is, they have the potential to receive many requests in the near future.

In our previous work, we introduced the Binary-Tree Based Estimation (BTBest) algorithm, which uses a binary tree structure to estimate the *potential* of a file, i.e., its increasing or decreasing demand for a period of one round. By predicting the increasing demand of files with high potential values, it makes them available sooner than other strategies do, so higher hit ratios are achieved, more file requests are completed and the job execution times are reduced.

In this paper, we extend our previous work and propose the CBTBest algorithm (Centralized BTBest), which uses a centralized approach that aims at minimizing the memory consumed and the number of computations needed for our binary tree based strategy. The rest of this paper is organized as follows: Section II briefly describes the system configuration. Section III presents shortly some necessary notations. Section IV presents the proposed scheme for data replication and the space and time analysis. In Section V we present our experimental results. Finally, section VI concludes the paper and offers aspects for future research work.

II. System Configuration

When a node requests a file f, it sends a request to its closest node. If the file is not found, the request is forwarded to the next closest node and so on. The grid consists of a number of clusters, each one having a number of cluster nodes, n and a header node H_i. Generally, the header nodes are responsible for managing site information, such as file sizes, distances and shortest paths between nodes, etc. [2].

In our centralized approach, the header nodes are also responsible for accepting information regarding the file requests for selected batches of files (this will be described in the next section), for estimating the file potential based on the information received and finally, and for preparing a list of the files to be replicated per node.

III. Notations

We divide the time in a set of rounds denoted by R. In turn, each round consists of r smaller *slots* denoted by t. During each slot, the *demand* $\mathcal{D}$ for a file in each node of the grid can either be increased by a factor x or decreased by a factor y, where $x \geq 1 > y \geq 0$. The sensitivity in user behavior is denoted by $B, B \in [0 \ldots 1]$. A value of B that approaches 1 shows almost constant behavior, while a value close to 0 indicates change in users' behavior. The number of requests for file f in node n during slot t after an increase in its demand by a factor x and after a decrease in its demand by a factor y, are denoted by $NR_t^{f,n}(x)$ and $NR_t^{f,n}(y)$, respectively.

Now, if we know the values of $NR_t^{f,n}(x)$ and $NR_t^{f,n}(y)$, we can estimate the value of $NR_{t-1}^{f,n}$. In other words, if we know the number of requests at slot t that is derived from either increasing or decreasing demand we can estimate $NR_{t-1}^{f,n}$, the number of requests in the previous slot. In our previous work [1], we showed that this value is

$$NR_{t-1}^{f,n} = \frac{xNR_t^{f,n}(y) - yNR_t^{f,n}(x)}{x-y} \tag{1}$$

The requests for a file during one round can be viewed as a random walk consisting of a succession of random steps, one step per slot, during which the demand is either increasing or decreasing. Since these steps are either x (increasing demand) or y (decreasing demand), they can be represented as paths of a binary tree, all the way from the root to a node at the bottom level of the tree. We will refer to $x-$steps, when the demand for a file during a slot has been increasing and to $y-$steps when the demand for the same file has been decreasing. A path can be fully described by its $x-$ and $y-$steps, e.g., a xyx path is a path starting with a $x-$step followed by a $y-$step, followed by a $x-$step while a $xxxyy$ path is a path starting with 3 successive $x-$steps followed by two $y-$steps.

IV. The CBTBest Strategy

The CBTBest strategy has five phases: (1) File Batching, (2) The binary tree construction in each cluster's header node, (3) Estimation of the node values of the tree, (4) Estimation of file potential, and (5) Computation of file popularities and decision regarding the replicas.

Phase 1: File Batching

In BTBest, each node stores its own tree structure and computes the potential and scope for the files for which it had requests. In CBTBest, there is only one binary tree structure per cluster and it is stored in the header node. Also, in BTBest, the potential and scope were estimated for each requested file. In the centralized version of BTBest, the potential and scope are estimated for batches of files, in the header node. The batches are formed in each node in the following way: From the information regarding the files requested from the previous round, each node creates groups of files in the form $< Batch\_Id, D_0, NR_0 >$, where $Batch\_Id$ is an identification for each file in the batch, D_0 is the file demand and NR_0 is the number of requests. Grouping is implemented with values for D_0 and NR_0 that lie in the same intervals which have already been predefined by each cluster, based on the file requests it receives.

For example, a batch including 5 files with demands 4,5,5,6,5 and number of requests 30,40,35,36,34, can be batched as $< b_1, 5, 35 >$. Similarly, a batch of files with demands 4,5,5,6,5 and number of requests 50,60,56,54, can be batched as $< b_2, 5, 55 >$. Each node is responsible to be aware of the files included in each batch.

Phase 2: Tree Construction

The initial construction of the tree for a file f, requires to set two threshold values T_x and T_y for the maximum value of x and the minimum value of y, respectively. The maximum increase in a file's demand for one slot [assume that $B \approx 1$] would be $D_t = D_{t-1}T_x$ and the maximum decrease would be $D_t = D_{t-1}T_y$. Thus, if the demand for a file keeps increasing then, its maximum value after r slots would be $D_t = D_{t-1}T_x^r$. Similarly, if the demand keeps decreasing then, its minimum value would be $D_t = D_{t-1}T_y^r$. With these demand values, we can compute the number of request values for the leafs of the tree. The leaf values form intervals, in the form $[NR_t^{f,n}(y), NR_t^{f,n}(x)]$. These intervals

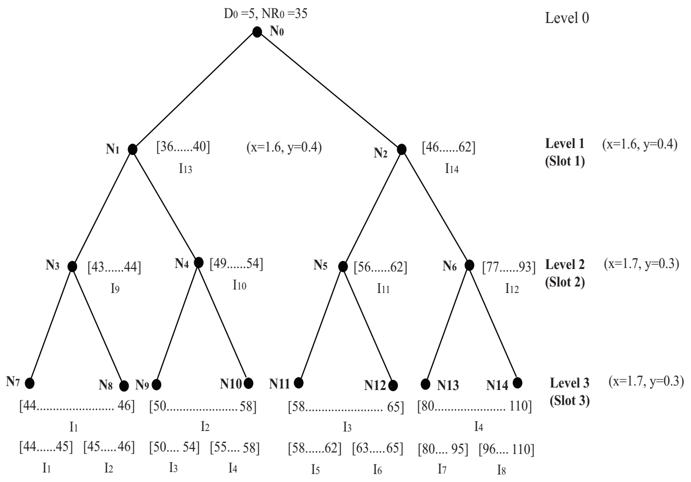

Fig. 1. A tree with the number of requests in the leaf nodes, where $NR_0 = 35$ and $D_0 = 5$.

are further halved, in order to create subintervals in the form $[min_t^{f,n}(y) \ldots min_t^{f,n}(x)]$ and $[max_t^{f,n}(y) \ldots max_t^{f,n}(x)]$. These are the subintervals that will be used in Phase 3 of the CBTBest. For example, consider Fig. 1: the tree structure has accepted as input a batch $< b_\mu, 5, 35 >$: There are eight leaf nodes that initially form four intervals $I_1 \ldots I_4 = [44 \ldots 46], [50 \ldots 58], [58 \ldots 65], [80 \ldots 110]$ (On top of the tree, the values of $NR_0 = 35$ and $D_0 = 5$ are added to give a value of 40 to node N_0). Then, these four intervals are halved, generating: $I_1 \ldots I_8 = [44 \ldots 45], [44 \ldots 46], [50 \ldots 54], [55 \ldots 58], [58 \ldots 62],$ $[63 \ldots 65], [80 \ldots 95], [96 \ldots 110]$ (bottom of Fig. 1).

Phase 3: Upper Node Values Estimation

Upon completing Phase 2, the header node estimates the remaining tree values using a bottom-up approach, under the hypothesis that the demand between slots $t-1$ and t may have increased by x with a probability p or decreased by y with a probability $(1-p)$. Each tree node stores two values, a minimum value $min_t^{f,n}$ and a maximum $max_t^{f,n}$, representing an interval $[min_t^{f,n} \ldots max_t^{f,n}]$. To compute a value for the immediately upper tree node, Eq. (1) is used two times, once to estimate the min value of the upper node interval, $min_{t-1}^{f,n}$ and one to estimate the max value of the upper node interval, $min_{t-1}^{f,n}$. So, (1) is rewritten as

$$min_{t-1}^{f,n} = \frac{x\ min_t^{f,n}(y) - y\ min_t^{f,n}(x)}{x-y} \tag{2}$$

$$max_{t-1}^{f,n} = \frac{x\ max_t^{f,n}(y) - y\ max_t^{f,n}(x)}{x-y} \tag{3}$$

In the example of Fig. 1, by using (2) and (3), we have obtained the intervals at the upper levels of the tree. The values of x and y are equal to 1.7 and 0.3 respectively for the last two slots and equal to 1.6 and 0.4 for slot 1, to illustrate the fact that the values of x and y may be slightly changing between slots.

Phase 4: Estimating the File Potential
In the fourth phase, the actual number of requests for each file in a batch are read at the end of the round and every possible path that leads to this value is examined. For each such path, the number X of $x-$steps and the number Y of $y-$steps are computed. Their difference form the metric we called *potential* $\mathcal{P}_{f,n}$ for f in node n after a round r.

For example, assume that the header node reads that the number of requests for file f_1 of the batch sent from node n is $NR_{f,n} = 80$. For reference, we also labeled all nodes with $N_i,\ i = 1 \dots 14$. There are two ways to reach an interval that includes 80:

(1) $N_0 \rightarrow N_2 \rightarrow N_6 \rightarrow$ one $y-$step: In this case, we have a $xx-$path (the path ends at level $k = 2$) leading to I_{12} where 80 lies. Thus, there was no request for f between slots 2 and 3 (that is, $y = 0$). We can simply add one more $y-$step to indicate no further requests for f after slot 2.

(2) $N_0 \rightarrow N_2 \rightarrow N_5 \rightarrow N_{12}$ (80 is $>$ than the maximum value of the interval $[63 \dots 65]$ located in N_{12}. In this case, we have a $xyx-$path. In total, the sum of $x-$steps is $X = 2 + 2 = 4$ and the number of $y-$steps is $Y = 1 + 1 = 2$. Thus $\mathcal{P}_{f,n} = X - Y = 4 - 2 = 2$.

Phase 5: File Popularity and Decision About the Replicas
The total popularity of a file f in node n is computed as follows:

$$FP_{f,n} = NR_{f,n} \times 2^{\mathcal{P}_{f,n}}, \tag{4}$$

where $NR_{f,n}$ is the number of requests for file f at the end of a round in node n while $\mathcal{P}_{f,n}$ is the potential of f during the same round in the same node. If $\mathcal{P}_{f,n} > 0$ then the number of requests is multiplied by a power of 2, if $\mathcal{P}_{f,n} < 0$ then the number of requests is divided by a power of two, and if $\mathcal{P}_{f,n} = 0$ then there is no change. Each node sorts the files in decreasing order with respect to the FP values (i.e., the file with the highest FP appears first) and selects a percentage of them to be replicated.

A. Space and Time Analysis

Space Analysis: Now, let us assume that μ batches of files are generated throughout the network and that two unsigned bytes are enough to store each of the values of D_0 and NR_0 and one unsigned byte is enough to store the batch id. Thus, the header node needs $5\mu \times N$ bytes to store the input from a network of N nodes, where clearly $\mu < F($ the total number of files). In Phases 2 and 3 of the CBTBest scheme, the header node generates a binary tree structure, computes all the tree node values and stores the tree into its memory. The tree is then repeatedly used for incoming batches to compute the potential. The size of the binary tree is *by no means related to the number of files* requested in the node. To compute the potential for each file, the header node uses the same tree structure with different batch input value. Thus, a maximum of $\mu 2^{r+1}$ values (the number of tree nodes) need to be stored. If each value needs 2 unsigned bytes then, $2 \times 2^{r+1}$ bytes are required. Phases 4 and 5 compute the file popularity and the node suitability. Phase 4 is the only phase dominated by the number of requested files F. The header node requires two bytes to store each potential value. The last phase is less memory-consuming since it involves only the files (assume they are F' in total) selected for replication. Normally, F' is a small percentage (10-20%) of F. In total, the CBTBest strategy needs $5\mu \times N + 2\mu \times 2^{r+1} + 2F + 2F' \approx 2F$ (since the number of nodes N is less than the number of requested files), which is 4 times less than the amount of memory required by the BTBest for the same computations.

Clock cycles		1	2	3	4	5	6	7	8	9	10
Segment	1	FB1	FB2	FB3	FB4	FB5	FB6	FB7			
	2		FB1	FB2	FB3	FB4	FB5	FB6	FB7		
	3			FB1	FB2	FB3	FB4	FB5	FB6	FB7	
	4				FB1	FB2	FB3	FB4	FB5	FB6	FB7

Fig. 2. Pipelining the CBTBest Strategy.

Time Analysis: Apparently, the major cost of the proposed scheme is the computation of the file potential. Practically, every round has to be divided into as many slots as possible, so that the scenarios created with different x and y values per slot or even with different probabilities are as many as possible. For example, if $r = 10$, then we can have as many as 1023 overlapped intervals, offering many different ways to reach a number of requests for f.

From the hardware point of view, the computations involved in CBTBest can be analyzed to the following sub-operations (SO): SO1: Input a batch of files, SO2: Binary tree construction and estimation of tree-node values, SO3: File potential estimation, and SO4: File popularity computation.

Clearly, each of the four sub-operations needs data from the previous one, so we can carefully organize them in such a way to execute them in a pipeline scheme. Each segment is responsible for one sub-operation and each file batch (denoted by FB) has to go through all stages to have all potentials computed. In Fig. 2 we show how to pipeline 7 file batches.

The total time cost of the CBTBest strategy is determined by the cost of the SOs. The first SO is a read operation that takes a trivial amount of time, since every input read is 5 bytes. The second SO requires $2^r \times r$ computations to estimate the values of the leaf nodes, $2^r/2 = 2^{r-1}$ computations to estimate the values of the nodes starting from level r to level $r-1$, another $2^{r-1}/2 = 2^{r-2}$ computations to estimate the values of the nodes starting from level $r-1$ to level $r-2$, and generally 2^{r-k} computations to estimate the values of the nodes from level $r-k+1$ to level $r-k$. These computations are $2^r \times r + 2^r - 1$ in total. The third SO is similar to a tree traversal, which can be performed in r steps per file, thus its

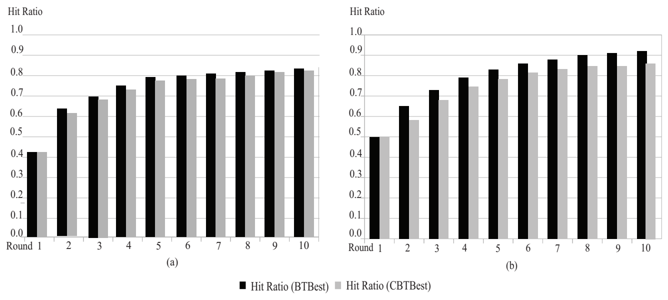

Fig. 3. Comparison of hit ratios between BTBest and CBTBest.

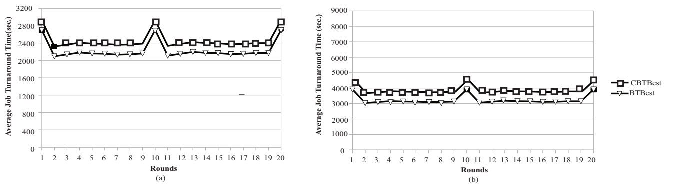

Fig. 4. Comparison of AJETs between BTBest and CBTBest.

cost is $F \times r$. The last SO requires exactly F' computations, one per file. Thus, the total computations per file for the first three SOs are $2^r \times r + 2^{r-1} + F \times r + F'$.

Clearly, the factor that dominates the total computation time is clearly $F \times r$, SO3, (since the number of requested files can grow larger, compared to the value of r). Thus, the time required by this sub-operation "absorbs" the time required by the others as the sub-operations are performed in parallel. Thus, the total computation cost is $F \times r$, much less than $\left[F \times \left(2^r \times r + 2^{r-1} + r\right) + F + F'\right]$ required by the BTBest strategy for the same computations.

V. Experimental Results

In our previous work, we compared the BTBest with state-of-the-art strategies like PFRF [2] and IPFRF [11] and we showed that the BTBest strategy outperforms PFRF and IPFRF in terms of the *hit ratio*, the number of requests completed at each round divided by the total number of request and in terms of the *average job execution time* (AJET). Here, we compare the BTBest strategy to the CBTBest strategy, to confirm that the newly proposed scheme can operate almost qually well while it consumes less memory and computational time. The simulation configuration is as described in [1].

To compare the hit ratios, we run two sets of experiments, similar as in our previous work: In the first set, each node replicates 20% of the files (Fig. 3(a)), the ones with the highest popularity. The hit ratio for BTBest and CBTBest increases quickly, reaching $\approx 75\%$ after one round and as more replications occur after consecutive rounds, it keeps growing. The BTBest strategy is a bit more efficient in terms of the hit ratio, around 1.5-2% on the average. In the second set, each node replicates 10% of the files (Fig. 3(b)). The behavior of the compared techniques is generally the same. However, the hit ratios are reduced by 7%-10% as a result of reducing the replicas and the BTBest strategy overcomes the CBTBest by about 5%, as a result of taking the average D_0s and NR_0s for inputs of the tree.

To compare the AJETs, we also run two simulation sets. In the first set (Fig. 4(a)), the file sizes varied from 100 MB-1GB, while in the second set (Fig. 4(b)), they varied from 1-2 GB. The other parameters were as in [1]. In both sets, the BTBest outperforms CBTBest by an average of 10%. There are two reasons behind these results: (1) the BTBest has higher hit ratio, and (2) the BTBest takes into account the *file scope*, so it offers better AJETs for jobs running on faraway nodes. The file scope metric has not yet been introduced in CBTBest.

VI. Conclusions

In this work, we have extended our previous work, the BTBest algorithm, and presented the Centralized BTBest. The CBTBest scheme is more cost-effective in terms of memory, because it uses just one tree structure stored in the header node for each cluster. Also, it is more cost-effective in terms of computational cost, because it uses a pipeline-based computational scheme. Experimental results have shown that the CBTBest strategy is almost as effective as BTBest in terms of hit ratio and AJET.

In our future work, we plan to develop a mixed BTBest and CBTBest strategy, to exploit the advantages of both schemes. Also, both schemes have to be tested in a real data grid and other contexts like HPC. Finally, their implementation to higher-dimensional trees with the insertion of more options regarding the increasing/decreasing factors is also of great interest.

References

[1] S. Souravlas and A. Sifaleras, "Binary-tree based estimation of file requests for efficient data replication," *to appear in IEEE Transactions on Parallel and Distributed Systems*, 2017.

[2] M.-C. Lee, F.-Y. Leu, and Y.-p. Chen, "PFRF: An adaptive data replication algorithm based on star-topology data grids," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1045–1057, 2012.

[3] K. Sashi and A. S. Thanamani, "Dynamic replication in a data grid using a modified BHR region based algorithm," *Future Generation Computer Systems*, vol. 27, no. 2, pp. 202–210, 2011.

[4] M. Bsoul, "A framework for replication in data grid," in *Proc. of the IEEE International Conference on Networking, Sensing and Control*, 2011, pp. 233–235.

[5] M. Bsoul, A. Al-Khasawneh, Y. Kilani, and I. Obeidat, "A threshold-based dynamic data replication strategy," *The Journal of Supercomputing*, vol. 60, no. 3, pp. 301–310, 2012.

[6] R.-S. Chang and H.-P. Chang, "A dynamic data replication strategy using access-weights in data grids," *The Journal of Supercomputing*, vol. 45, no. 3, pp. 277–295, 2008.

[7] N. Mansouri and G. H. Dastghaibyfard, "A dynamic replica management strategy in data grid," *Journal of Network and Computer Applications*, vol. 35, no. 4, pp. 1297–1303, 2012.

[8] L. M. Khanli, A. Isazadeh, and T. N. Shishavan, "PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 233–244, 2011.

[9] Z. Wang, T. Li, N. Xiong, and Y. Pan, "A novel dynamic network data replication scheme based on historical access record and proactive deletion," *The Journal of Supercomputing*, vol. 62, no. 1, pp. 227–250, 2012.

[10] J.-J. Wu, Y.-F. Lin, and P. Liu, "Optimal replica placement in hierarchical data grids with locality assurance," *Journal of Parallel and Distributed Computing*, vol. 68, no. 12, pp. 1517–1538, 2008.

[11] M. Bsoul, A. E. Abdallah, K. Almakadmeh, and N. Tahat, "A round-based data replication strategy," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 31–39, 2016.