Binary-Tree Based Estimation of File Requests for Efficient Data Replication

Stavros Souravlas, *Member, IEEE,* and Angelo Sifaleras

***Abstract*—Recently, data replication has received considerable attention in the field of grid computing. The main goal of data replication algorithms is to optimize data access performance by replicating the most popular files. When a file does not exist in the node where it was requested, it necessarily has to be transferred from another node, causing delays in the completion the file requests. The general idea behind data replication is to keep track of the most popular files requested in the grid and create copies of them in selected nodes. In this way, more file requests can be completed over a period of time and average job execution time is reduced. In this paper, we introduce an algorithm that estimates the potential of the files located in each node of the grid, using a binary tree structure. Also, the file scope and the file type are taken into account. By potential of a file, we mean its increasing or decreasing demand over a period of time. The file scope generally refers to the extent of the group of users which are interested or potentially interested in a file. The file types are divided into read and write intensive. Our scheme mainly promotes the high-potential files for replication, based on the temporal locality principle. The simulation results indicate that the proposed scheme can offer better data access performance in terms of the hit ratio and the average job execution time, compared to other state-of-the-art strategies.**

***Index Terms*—Data Replication, Binary Trees, Data Grid, File Popularity**

I. Introduction

Many applications are moving towards a distributed interconnected environment. In this environment, the data storage and all computational resources are distributed throughout different and widespread locations. A Data Grid is such an environment.

A Data Grid can have a huge number of users that need to have access to huge data volumes. For example, consider a set of documents that needs to be read and processed by a number of coauthors spread worldwide, in a distributed way. The access to huge data volumes by huge number of users can be very time consuming. As the size of the system is increased, the task of providing such data services becomes more difficult since its users suffer from long delays in data access.

In such environments, data replication is required so that the users can retrieve the requested data from storage residing in nearby nodes [1], [2], [3], [4]. The performance of the distributed environment is crucially affected by the replication strategy used. The vast majority of the known replication strategies determines the replicas [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] by computing a simple metric based on the number of requests for each file. The "hottest" files, i.e, the ones with the highest metric value, are selected for replication due to the temporal locality principle. This principle states that it is quite possible that, the files with high recent demand will be requested again soon, with even higher rates. Since the computations are quite simple, the strategies mainly focus on the problem of selecting the most suitable nodes for storing the replicas. The three main drawbacks of the strategies proposed are related to the following issues:

1. *Changes in user behavior:* The schemes presented in the literature do not take into account the changes that might occur in the interest of users for certain files [16]. Instead, they are mostly interested in one or more factors that determine the importance of the file themselves, like the file size, the number of requests for a file or the contents of a file [6], [8], [9], [12], [17], [14].
2. *File Potential:* The strategies so far consider the number of requests as the major metric for computing the popularity of each file. However, taking an absolute value of requests as a metric may lead us to misleading estimations regarding the importance of a file. As an example, consider a file f_1 that was requested 100 times over a period of time t_1 and after some more periods, let us say at time t_r, the total number of requests for the same file is 105. The same computations for another file f_2 are 20 at t_1 and 75 at t_r. Taking the absolute values 105 and 75 as metrics, the popularity of f_1 would be higher compared to the popularity of f_2, ignoring the fact that the demand for f_2 has tripled and it is expected to have much more requests in the near future. Thus, f_2 should probably deserve to be ranked higher in the list of candidates for replication, compared to f_1.

The first of the above issues is addressed by dividing the time into time intervals called *rounds* [16], [18] and keeping track of the file requests in each of these rounds. This information gives us a picture regarding the most preferred files in each node and can reflect any change in user behavior over time. The *file popularity* computed for every file in every node is the metric used to register or estimate the current and future user behavior.

The second issue has been partly addressed in two papers [9], [14]. The general idea is as follows: The file popularity is determined by dynamic weights and history access log files. If t periods have passed, the earliest period is assigned a weight of 2^{1-t}, the next to earliest is assigned a weight of 2^{2-t}, and the most recent is assigned a weight of $2^{t-t} = 1$. Thus, this weight factor is halved between successive periods.

Stavros Souravlas and Angelo Sifaleras are with the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece.
Email: sourstav, sifalera@uom.gr

Please cite this paper as:
Souravlas S. and Sifaleras A., Binary-tree based estimation of file requests for efficient data replication, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 7, pp. 1839-1852, 2017.
The final publication is available at IEEE Xplore via http://dx.doi.org/10.1109/TPDS.2017.2650228

The popularity of a file f over t periods in a node n will be the sum of all products formed by the weight factors and the number of accesses of f during the corresponding period. For the previous example, the popularities of f_1 and f_2 will be $100 \times 2^{-1} + 105 \times 2^0 = 155$ and $20 \times 2^{-1} + 75 \times 2^0 = 85$, respectively. However, these two techniques have two weaknesses:

1. The first weakness is apparent from the computations we just described. Normally, we would expect a file with higher increase rate like f_2 to have higher popularity, but this is not the case. This means that the popularity is highly affected by the latest access value, although the other periods are also taken into account.
2. The second weakness is the memory consumed. To compute the popularity, each site maintains an access log file for each separate file, that keeps the number of file accesses in a *header* form, $\langle$*FileId, ClusterId, Number*$\rangle$. All the headers from all the nodes are grouped in a cluster header, and the information from the cluster header is sent to other cluster headers. Assuming that an access value can be stored in 2 unsigned bytes, then each node needs $2F$ bytes per interval only to store the number of accesses for each file.

The contribution of this work is twofold: first, it promotes the high-potential files for replication, thus it makes them available sooner that they would be, based only on their access numbers. Second, it combines the potential with the file scope to promote for replication those files that have growing demand and are requested by a broader group of users. Our strategy uses a complete binary-tree based mechanism used for the potential computation of all files, thus it is less memory-consuming, and it can be fully parallelized, so that its computations are performed in the minimum possible time.

The rest of this paper is organized as follows: Section 2 summarizes the related work. Section 3 describes our strategy. Section 4 presents the simulation configuration to verify that the simulation results obtained comply to the theoretical presentation of our scheme. Section 5 discusses the simulation results. Finally, Section 6 concludes the paper and offers aspects for future research work.

II. Related Work

In the recent years, a lot of effort has focused on the problem of data replication in a Data Grid. In [9], a dynamic data replication mechanism called Latest Access Largest Weight (LALW) was proposed. LALW associates different weights to historical data access records, thus differentiating the importance of each record. The metric used to represent the importance of access histories in different time intervals was called *Access Frequency (AF)* and it was based on the number of requests for each file over these time intervals. The algorithm does not have a mechanism to decide exactly which node will accommodate each replica. The LALW mechanism addresses the problem of limited storage available in each node, by deleting the least recently used files. This algorithm does not determine the node within the cluster where each popular file will be replicated and it only considers for replication the files with large number of requests, regardless of their potential or scope.

There are strategies which are based on the notion of *spatial locality*. The spatial locality states that a file related to recently accessed files will probably be requested soon. In [10], a Predictive Hierarchical Fast Spread (PHFS) technique was proposed. Based on spatial locality, PHFS tries to predict the future needs and pre-replicates the proper files in a hierarchical manner to increase access locality. Initially, the scheme collects the information regarding accesses from the whole system and creates log files about them. Then, it uses data mining techniques such as association rules and clustering to get the proper information, e.g., groups of files accessed together with high probability, and the most frequent access patterns. These files are assumed to have logical spatial locality. In [14], the authors proposed a data replication algorithm based on historical access record to find files that were requested at about the same time. These files are assumed to have a relationship and thus, they are selected for replication. Algorithms based just on spatial locality have a drawback: they cannot determine files with rapidly growing demand if a kind of close relationship is not revealed (e.g., if a number of request for the files of a scientific project is followed by a couple of requests for a news article).

The majority of the well-known strategies are based on the notion of temporal locality described in the previous section. In [6], the authors presented a dynamic replication strategy that takes into account the number and frequency of requests, as well as other factors such as the size of the replica and the last time the replica was requested. The proposed scheme, EFS (Enhanced Fast Spread), keeps only the important replicas and replaces the less important replicas with others, more important. Sashi and Thanamani [17] presented a Modified Bandwidth Hierarchy Replication (BHR) algorithm, to overcome the limitations of the standard BHR [19] algorithm. BHR replicated the popular files *as many times as possible* within a region, while the enhanced BHR tried storing replicas in the site where the file has been accessed most frequently, assuming that it will be requested again in the near future. In [12], a Dynamic Hierarchical Replication algorithm, DHR, was proposed. DHR places each replica in the best site that has the highest number of requests for the specific replica assuming that more requests for the same file will arrive shortly. In [20], the authors proposed a novel scheme that, incurs no communication overhead between the users and the servers upon job arrival. The approach is based on creating a number of replicas of each job and sending these replicas to different servers. These replicas are assumed to satisfy future requests on these servers, based on temporal locality. Other schemes also employ threshold values to have some type of control over the number of replicas [7], [8]. In addition, the authors in [8] proposed a scheme that places the replicas found in each node in a certain category. Each category takes a value that determines the importance of each file for each node. When the node starts running out of storage it stores only the replicas that belong to the most important category or categories, depending on the available space left. The temporal-locality-based strategies mentioned consider only the

number of requests to determine if a file should be replicated. Factors such as the file potential or scope are not considered at all. Also, they consider that the user behavior is generally fixed.

The changes in user behavior were studied by dividing the total time into smaller periods and studying each of these periods separately: In [18], the authors proposed the PFRF (Popular File Replicate First) strategy, which divides the time into *rounds* and computes the file popularity at the end of each round. Then, it replicates a percentage of the files, which are found to be more important. Again, the main problem with this strategy is that it is based only on the number of requests to determine the file popularity. Also, there is no strategy that determines the nodes where each replica is going to be placed. These drawbacks were partially dealt with in an interesting paper by Bsoul et. al. [16]. In their technique named Improved PFRF (IPFRF), the file popularity is determined not only by the number of requests but also by the file size. The choice of this factor is due to the limited storage. Also, the technique provides a method to compute the *suitability* of each node inside a cluster. A higher suitability indicates if a node is suitable to place a replica of a specific file. This suitability metric is determined by three factors $A, B,$ and C that sum to 1 and can have different weights depending on the user needs. A is a weight assigned to the number of requests for the specific file submitted in a node, B is the weight used to increase the suitability of nodes that will assure load balancing over the network when transferring the replicas, and C is a weight used to increase the suitability of the central nodes.

This work uses a complete binary-tree based estimation (BTBest) technique that estimates the potential and the scope of each file, to promote files with rapidly growing demand. The potential and the scope are used as factors to compute the popularity of each file. The files are separated into read and write intensive, because write intensive files should be sparingly replicated, as more replicas would require more data maintenance costs, thus the strategy becomes less efficient. As in [16], our strategy also computes the suitability of each node as a storage for a replica. The factors that we consider in this computation is the distance between the particular node and the source of the replica, the number of requests for the replicated file submitted in the particular node, and the potential of the file. The simulation results show that, our strategy can increase the ratio of completed file requests to the total number of requests (hit ratio) and the average job execution time; thus increasing the data access performance.

III. BTBest Strategy

BTBest is a strategy that divides the time into rounds as in [16], [18], and further, it divides each round in smaller time units called *slots*. Based on a complete binary tree structure, it estimates the *potential* of a file in a round, using a bottom up approach. The potential of a file is used as a factor for computing the file popularity. Before presenting BTBest in detail, we provide a motivating example to facilitate the discussion that follows.

A. A Motivating Example

Assume that, at time t_1, the number of requests for a file f_1 located in a node n is 55 and that there have been 5 requests for this file between time t_0 and t_1, in other words, the demand for f was 5 during this time interval and the number of requests at time t_0 was 50. Also, assume that the demand for f_1 is doubled at discrete times $t_2, t_3,$ and t_4. That is, the demand will be $5 \times 2 = 10$ at t_2, $10 \times 2 = 20$ at t_3 and $20 \times 2 = 40$ at t_4. Thus, the total number of requests at time t_4 will be 50+5+10+20+40=125.

Now, assume that, for a second file f_2, the number of requests that have arrived to the same node n is 80 and that there have been 5 requests for this file between time t_0 and t_1, in other words, the demand for f_2 was 5 during this time interval and the number of requests at time t_0 was 75. Also, assume that the demand for f_2 is reduced by 20% at time t_2 and by 50% at t_3, t_4, so it will become $5 \times 0.8 = 4$ at t_2, $4 \times 0.5 = 2$ at t_3 and $2 \times 0.5 = 1$ at t_4. Thus, the total number of requests at time t_4 will be 75+5+4+2+1=87.

The question that arises is "what will be the replication policy at time t_1?" or, in other words, "what is the popularity of the two files at time t_1?" Considering only the number of requests, the decision will favor f_2 since there have been 75 requests for f_2 and only 50 for f_1. However, the demand for f_1 keeps growing while the demand for f_2 keeps falling off, meaning that from the potential point of view, f_1 should be favored. This is shown in Figures 1(a) and 1(b). An algorithm that considers only the number of requests to make a replication decision will replicate f_1 after t_3, when the number of requests for f_1 becomes higher compared to the number of requests for f_2. An algorithm that takes the potential into account would replicate f_1 just after t_1 since its potential is high and it is expected to have a high number of requests in the near future, due to temporal locality. This will improve the performance of grid jobs.

Note that, we may ask an additional question: How can we protect ourselves from creating replicas for files that present high potential only for a period of time? Such files may, for example, include data shared by a small group of students working on a project. The members of this group request this file for a period of time (thus increasing its potential), but this file is of small or no interest to anyone else. For such cases, the scope of the file should also be taken into account, in combination with the potential. If a percentage of the requests for a file is submitted to node n from faraway nodes, the scope of this file increases due to the *geographical locality* [10]: files accessed by users in a part of the grid are likely to be requested by their neighbors as well. So, when the distance of the submitted requests is larger, the scope of the file becomes larger too and the number of potential users increases.

In our previous example, if the increasing demand for f_1 comes from nodes of the same cluster as n, then the replication strategy should be cautious regarding the replication of f_1. On the contrary, if the increasing demand comes from faraway nodes residing in other clusters of the grid, then the replication strategy should replicate f_1 with high confidence.

In the remaining of this section, we describe the BTBest

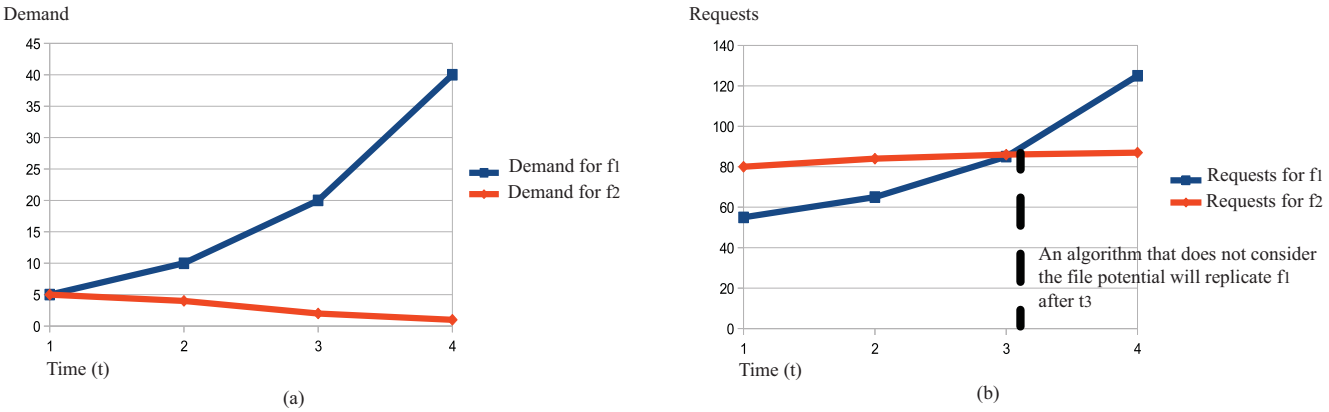

Fig. 1. (a) Increasing/Decreasing demand for files f_1 and f_2, correspondingly. (b) The total number of requests for f_1 becomes higher than the number of requests for f_2 after t_3.

strategy focusing mainly on the topics discussed above. Initially, we give the mathematical background and some necessary notations. Then, we describe how BTBest estimates the potential and the scope of a file, in order to find its popularity. Finally, we discuss the node suitability, and perform time and space complexity analysis.

B. Background and Notations

The variables that will be henceforth used in this paper are listed in Table I. To study the changes in user behavior, we divide the total time in a set of rounds. Each round consists of r small intervals called *slots*. As seen in the example of the previous paragraph, during each slot, the *demand* $\mathcal{D}$ for a file in each node of a cluster can either be increased by a factor x or decreased by a factor y, where the increasing factor x and the decreasing factor y are such that:

$$x \geq 1 > y \geq 0, \tag{1}$$

The demand for a file follows a multiplicative, binomial process during a number of discrete periods. Thus, there are two possible values:

$$\mathcal{D}_t^{f,n} = \begin{cases} \mathcal{D}_{t-1}^{f,n} xB, & \text{for increasing demand} \\ \mathcal{D}_{t-1}^{f,n} yB, & \text{for decreasing demand} \end{cases} \tag{2}$$

where t is a time slot, $0 \leq t < r$ and B is a factor that represents the *sensitivity* in any change in user behavior. In our model $B \in [0 \dots 1]$, where the higher values of B indicate constant user behavior towards a file while the lower values indicate changes in user behavior towards the same file. We can consider B as a measure for the *temporal locality principle* [21], which states that the popular files in the past will be accessed more frequently than the others in the near future. The way that, the possible changes of B affect the simulations results, will be afterward presented in Section 4.

The demand for a file is always a positive value. We use x or y to represent an increasing or decreasing demand over a period of one slot (two variables, one for each case). To clarify this, let us return to the example of the previous paragraph. Assume that the demand for f_1 is quadruplicated in t_2 and halved at t_3: Then, from (2), $D_{t_2}^{f,n} = D_{t_1} \times 4 \times 1 = 20$ ($x = 4$) and $D_{t_3}^{f,n} = D_{t_2} \times \frac{1}{2} \times 1 = 20 \times \frac{1}{2} = 10$ ($y = 1/2$). This means that, there will be 20 requests from t_1 to t_2 and another 10 requests from t_2 to t_3. So, at t_3, the number of requests for f_1 will be 50+20+10=80.

TABLE I
LIST OF VARIABLES USED IN THIS PAPER

Parameter	Description
R	A set of rounds
$D_t^{f,n}$	Demand for file f in node n during slot t
D_{path}	The demand produced after a file has followed a path of increasing and/or decreasing *steps*, e.g., D_{xyx}
r	Number of slots per round, between 1 and 10
t	A time slot, $t \in [1 \dots r]$
f	A file
B	Sensitivity factor
$NR_t^{f,n}$	Number of requests for file f in node n during slot t
$NR_t^{f,n}(x)$	Same as previous, when it has increased by x
$NR_t^{f,n}(y)$	Same as previous, when it has decreased by y
$NR_{f,n}$	Number of requests for f at the end of a round.
x, y	Factors for increasing, decreasing demand
p	Probability of increasing demand from slot to slot
T_x, T_y	Threshold values for x and y, respectively
$\mathcal{P}_{f,n}$	Potential of a file f in node n after a round.
$\mathcal{P}_f$	Potential of a file for all nodes after a round.
$FP_{f,n}$	File popularity for a file f in node n
w, z	Weight factors to compute node suitability
$\delta_{n,j}$	Distance between nodes n and j (measured in hops)
$\Omega_{n,j}$	Distance between nodes n and j (expressed in probabilities)
$S_{f,n}$	Scope of a file f in node n
W	Percentage of write requests for a file

Using (2), we can compute the number of requests $NR_t^{f,n}$ for a file f located in a node n, during a time slot t:

$$NR_t^{f,n} = NR_{t-1}^{f,n} + \mathcal{D}_t^{f,n} \tag{3}$$

Now, let us set p, the probability of increasing demand during one slot; thus $(1-p)$ is the probability of decreasing demand. So, we can express the number of requests $NR_t^{f,n}$, during one time slot t, in terms of probabilities, using the following equation:

$$NR_t^{f,n} = pNR_{t-1}^{f,n}(x) + (1-p)NR_{t-1}^{f,n}(y) \tag{4}$$

where $pNR_{t-1}^{f,n}(x)$ and $(1-p)NR_{t-1}^{f,n}(y)$ are the numbers of requests for file f in node n during slot t after an increase

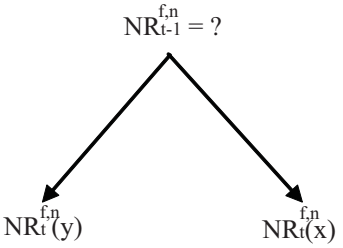

Fig. 2. Estimating $NR_{t-1}^{f,n}$ based on the values of $NR_t^{f,n}(x)$ and $NR_t^{f,n}(y)$.

in demand by a factor x and after a decrease in demand by a factor y, respectively. The probability of increased demand is p and the probability of decreased demand is $1-p$.

Now, let us express $NR_t^{f,n}(x)$ and $NR_t^{f,n}(y)$ in terms of the number of requests during a time slot, the demand, and the values of x and y. We have:

$$NR_t^{f,n}(x) = BxD_{t-1} + NR_{t-1}^{f,n} \quad (5)$$

$$NR_t^{f,n}(y) = ByD_{t-1} + NR_{t-1}^{f,n} \quad (6)$$

Now, assume that we know the values of $NR_t^{f,n}(x)$ and $NR_t^{f,n}(y)$ and we try to estimate the value of $NR_{t-1}^{f,n}$. In other words, we know the number of requests at slot t, that is derived from either increasing or decreasing demand and we try to estimate $NR_{t-1}^{f,n}$, as it is shown in Fig.2. Using (5) and (6) we have $NR_t^{f,n}(x) - NR_t^{f,n}(y) = BxD_{t-1}^{f,n} + NR_{t-1}^{f,n} - ByD_{t-1}^{f,n} - NR_{t-1}^{f,n}$ or $NR_t^{f,n}(x) - NR_t^{f,n}(y) = BxD_{t-1}^{f,n} - ByD_{t-1}^{f,n}$, from which we easily see that

$$B = \frac{NR_t^{f,n}(x) - NR_t^{f,n}(y)}{D_{t-1}^{f,n}(x-y)} \quad (7)$$

From (5), we see that $NR_{t-1}^{f,n} = NR_t^{f,n}(x) - BxD_{t-1}^{f,n}$. Now, after substituting B with the value found in (7), we get

$$\begin{aligned} NR_{t-1}^{f,n} &= NR_t^{f,n}(x) - BxD_{t-1}^{f,n} \\ &= NR_t^{f,n}(x) - \left[\frac{NR_t^{f,n}(x) - NR_t^{f,n}(y)}{D_{t-1}^{f,n}(x-y)}\right] xD_{t-1}^{f,n} \\ &= \frac{xNR_t^{f,n}(y) - yNR_t^{f,n}(x)}{x-y} \end{aligned} \quad (8)$$

Thus, we have derived an equation that gives us the number of requests for the previous time slot. This equation will be used in Stage 1 that is described next.

C. Stage 1: Estimating the Potential of a File

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. We will be referring to a $x-$step, and a $y-$step when the demand for a file during this step has been increasing or decreasing, respectively. Since the steps are either x or y, they can be represented as paths of a binary tree, all the way from the root to a node at the last level of the tree. 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 $x3y2$ or $xxxyy$ path is a path starting with 3 successive x-steps followed by two $y-$steps.

The BTBest algorithm estimates the potential of a file f, residing in a node n, in three phases:

1. In the first phase, it creates the binary tree and computes the values of the leaf nodes.
2. In the second phase, it works bottom-up on the tree, to estimate the upper node values.
3. In the third phase, it reads the actual number of requests for a file f at the end of the round and examines all possible paths on the tree that may lead to this value. These paths will be used to estimate the potential of the file.

In the remaining of this subsection, we will describe the workflow of these three phases to expose the ideas used by BTBest to estimate the potential and then, we will typically describe Stage 1.

Phase 1: Constructing the Tree and Estimating the Leaf Nodes

To construct the tree for a file f, we initially set two threshold values, T_x and T_y, to be the maximum value of x and the minimum value of y during a round, respectively. The existence of these thresholds can be justified by the fact that, the demand for a file during a time slot cannot increase or decrease infinitely. For example, in [18], the number of rounds was set to 20 and the duration per round was 1600s. Also, in the experiments conducted, the maximum number of requests for the most popular file in all rounds was found to be 175. So, it is rational to place a bound on the changes of demand during a time slot, a subdivision of a round. There are two ways to set the threshold values: Firstly, in an arbitrary way, e.g., $T_x = 4$ and $T_y = 0.5$, indicating that the demand for all files cannot increase more than a factor of 4 (quadruplication) or decrease by a factor less than 0.5, from slot to slot, and secondly, by recording demand values from previous rounds and set T_x and T_y to the maximum recorded increase and T_y decrease, respectively. The root value is set to the demand for f during the previous round, D_0. An example tree is shown in Fig. 3, where the root value is set to $D_0 = 5$ and the thresholds are $T_x = 2$, and $T_y = 0.5$.

From the tree of Fig. 3(a), it can be easily seen that each level of the tree represents a time slot. In each slot, the maximum increase or decrease in a file's demand is the demand found in the previous slot multiplied by T_x or T_y, respectively. These can be expressed by $D_t^{f,n} = D_{t-1}^{f,n} T_x$ and $D_t^{f,n} = D_{t-1}^{f,n} T_y$.

The leaf nodes represent the marginal values that appear when a path is taken along the tree, with an initial demand value D_0 (root) and threshold values T_x and T_y. In the example tree of Fig. 3, if the demand keeps increasing with the maximum increasing factor T_x, after $r = 3$ slots it will become $5 \times 2 \times 2 \times 2 = 5 \times 2^3 = 40$. If it increases by T_x for two rounds and then decreases by T_y for one round, it will become $5 \times 2 \times 2 \times 1/2 = 10$. If it keeps decreasing with the maximum decreasing factor T_y, after $r = 3$ slots it will become $5 \times 1/2 \times 1/2 \times 1/2 = 0.625$. The leaf node values are

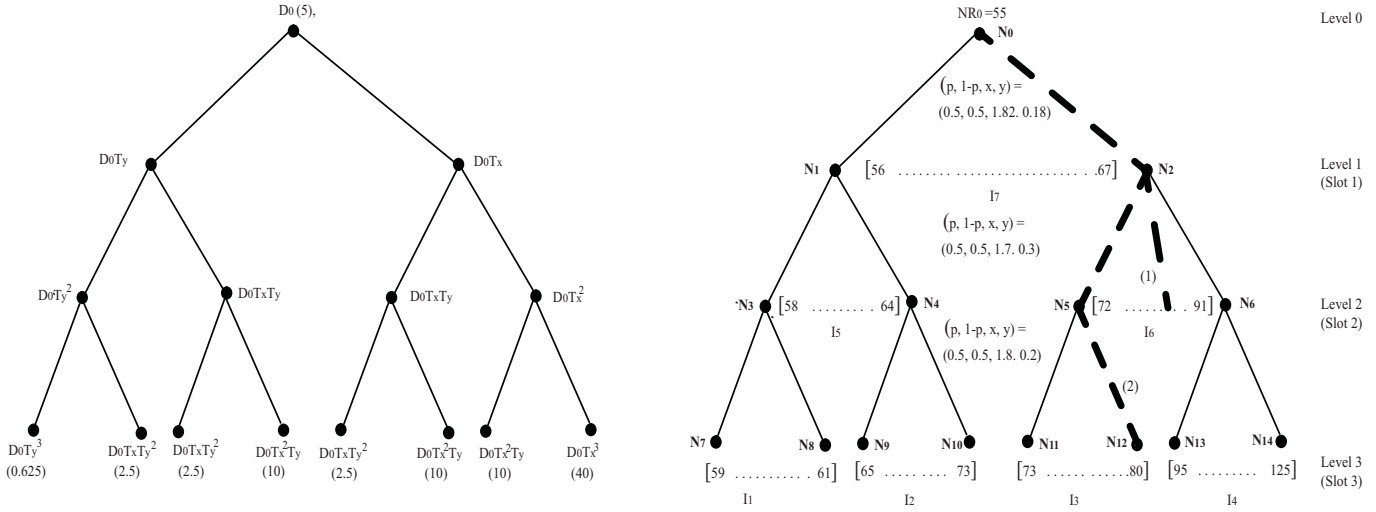

Fig. 3. (a) A tree that shows the maximum increase and decrease of demand for a file during 3 slots, $D_0 = 5,\ T_x = 2$, and $T_y = 0.5$, (b) A tree with the number of requests in the leaf nodes, where $NR_0 = 55$

shown in parentheses in Fig. 3(a). Note that, there are nodes having the same values. Specifically, the paths $yyx,\ yxy$, and xyy have the same demand value (2.5), while the paths $xxy,\ xyx$, and yxx have the same demand value (10).

More typically, we can use the following equations to compute the leaf node values:

$$D_{yr} = D_0(T_y)^r \quad (9)$$
$$D_{yr-1x} = D_0 T_x (T_y)^{r-1} \quad (10)$$
$$D_{yr-2x} = D_0 (T_x)^2 (T_y)^{r-2} \quad (11)$$
$$\dots$$
$$\dots$$
$$D_{xr-1y} = D_0 T_y (T_x)^{r-1} \quad (12)$$
$$D_{xr} = D_0 (T_x)^r, \quad (13)$$

where D_{path} is the demand produced after a file has followed a path of increasing and/or decreasing *steps*.

With the above computations and using (3) we can find the number of requests corresponding to the leaf nodes. For example (see also the motivating example in the beginning of this section), if the demand keeps doubling for three slots, there will be 125 requests for file f. For the xxy path (two increases by $T_x = 2$ followed by one decrease by $T_y = 0.5$), there will be 55+10+20+10=95 requests. Every pair of leaf nodes forms an interval I_j. For example $I_4 = [95 \dots 125]$. In our example, we have a total of $2^r/2 = 2^3/2 = 4$ intervals, $I_1 - I_4$. These intervals are shown in the bottom of Fig.3(b) and represent the maximum and minimum request values for a file f during a round.

The intervals computed in this stage will be used to estimate the node values at the upper levels of the tree, using a bottom-up approach. The other details of Fig.3(b) will be described in the subsequent paragraphs.

Phase 2: Estimating the Upper Node Values

From (8), we know that the number of requests in every round is related to the values of x and y. Particularly, (8) states that we can estimate the number of requests at slot $t-1$, if we know the number of requests at slot t and the values of x and y. When we try to estimate values using a bottom-up approach, we use 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 $(1-p)$. Thus, a relationship between x, y and their corresponding probabilities should be found. Proposition 1 proves the existence of such a relationship and typically defines it.

Proposition 1: The probabilities p and $(1-p)$ are related to x and y.

Proof: Let us return to Fig.2 and assume that all we know is that if $NR_{t-1}^{f,n}$ had been increased by a factor x, its new value would have been $NR_t^{f,n}(x)$. Similarly, if it had been decreased by a factor y, its new value would have been $NR_t^{f,n}(y)$. When we work bottom-up, we don't know if $NR_{t-1}^{f,n}$ has been the result of increasing or decreasing demand from the previous slot. Thus, every value $NR_t^{f,n}$, can also be expressed in terms of its previous value $NR_{t-1}^{f,n}$ as follows:

$$NR_t^{f,n} = NR_{t-1}^{f,n} + BD_{t-1}^{f,n} \quad (14)$$

Now, let us expand (14) using (7) and (8). Then, $NR_t^{f,n}$ is written as:

$$\begin{aligned} NR_t^{f,n} &= NR_{t-1}^{f,n} + BD_{t-1}^{f,n} \\ &= \frac{xNR_t^{f,n}(y) - yNR_t^{f,n}(x)}{x-y} \\ &+ \left(\frac{NR_t^{f,n}(x) - NR_t^{f,n}(y)}{D_{t-1}^{f,n}(x-y)} \right) D_{t-1}^{f,n} \\ &= \frac{xNR_t^{f,n}(y) - NR_t^{f,n}(y)}{x-y} \end{aligned}$$

$$+ \quad \frac{NR_t^{f,n}(x) - yNR_t^{f,n}(x)}{x-y}$$
$$= \quad \frac{NR_t^{f,n}(y)(x-1) + NR_t^{f,n}(x)(1-y)}{x-y} \qquad (15)$$

Now, we set $p = \frac{1-y}{x-y}$, thus $1-p = \frac{x-1}{x-y}$. Thus, (15) is written as

$$NR_t^{f,n} = pNR_{t-1}^{f,n}(x) + (1-p)NR_{t-1}^{f,n}(y)$$

which is (4). Thus, we have expressed the probabilities of increasing or decreasing demand in terms of x and y. ■

To summarize:

$$p = \frac{1-y}{x-y} \qquad (16)$$
$$q = \frac{x-1}{x-y} \qquad (17)$$

Equations (16) and (17) are interpreted as follows: Given that the number of requests for a file f in node n after slot t, is either $NR_t^{f,n}(x)$ or $NR_t^{f,n}(y)$ we estimate that the number of requests for the same file at the end of slot $t-1$ was NR_{t-1} and there is:

(a) a probability p that the demand has been increased by x during $t-1$, producing $NR_t^{f,n}(x)$.
(b) a probability $(1-p)$ that the demand has been decreased by y during $t-1$, producing $NR_t^{f,n}(y)$.

Let us return to Fig.3(b) and see how the BTBest strategy works bottom-up to estimate the node values for the upper levels of the tree. For these estimations, we use any values of x and y close to T_x and T_y respectively, such that there is a probability p for a $x-$step and a probability $(1-p)$ for a $y-$step. An obvious choice would be to set $p = 0.5$, so from (16) and (17), one can see that $x + y = 2$. So, if we choose equal probabilities between $x-$ and $y-$steps, we can use any values of x, y such that $x + y = 2$. For example, in Fig.3(b), we used $x = 1.8$ and $y = 0.2$ for the nodes of level 2, $x = 1.7$ and $y = 0.3$ for the nodes of level 1, and $x = 1.82$ and $y = 0.18$ for the nodes of level 0. The reason we make only small changes of x and y between consecutive steps is to avoid dramatic changes per slot.

For example, consider the nodes at level 2 (slot 2), $N_3 - N_6$. For this level, the algorithm has set $(p, 1-p, x, y) = (0.5, 0.5, 1.7, 0.3)$. Using (8) we get
$(1.8 \times 59 - 0.2 \times 61)/(1.8 - 0.2) = 59,$ for N_3
$(1.8 \times 65 - 0.2 \times 73)/(1.8 - 0.2) = 64,$ for N_4
$(1.8 \times 73 - 0.2 \times 80)/(1.8 - 0.2) = 72,$ for N_5
$(1.8 \times 95 - 0.2 \times 125)/(1.8 - 0.2) = 91,$ for N_6

Thus, two more intervals are created, $I_5 = [58 \ldots 64]$ and $I_6 = [72 \ldots 91]$. The other values of the tree are estimated in a similar way.

Phase 3: Estimating the File Potential

In the third phase, the BTBest strategy reads the actual number of requests for a file f at the end of the round and examines every possible path that leads to this value. For each such path, it computes the number of $x-$steps, X and the number of $y-$steps, Y. Their difference forms the metric called *potential* $\mathcal{P}_{f,n}$ for f in node n after a round r.

In the example of Fig.3(b), we have located these paths for $NR_{f,n} = 83$ (the number of requests for f at the end of this round, $NR_{f,n}$). Then, we try to locate every possible path that leads to an interval where $NR_{f,n} = 83$ lies. These paths are labeled by (1) and (2) on the tree of Fig.3(b) and they are depicted with dashed lines. There are two scenarios to end up in an interval that includes 83:

(1) $N_0 \rightarrow N_2 \rightarrow$ a $x-$step: In this case, we have a $xx-$path (the path ends at level 2) leading to I_6 where 83 lies. Thus, there was no request for f between slots 2 and 3 ($y = 0$). We can simply add one more $y-$step to indicate no further requests for f after slot 2.
(2) $N_0 \rightarrow N_2 \rightarrow N_5 \rightarrow$ a $x-$ step. In this case, we have a $xyy-$path leading to a value higher 80 (the value of N_{12}).

In total, the sum of $x-$steps in the above paths is the $X = 2 + 2 = 4$ and the number of $y-$steps is $Y = 1 + 1 = 2$. Thus the potential for f is $\mathcal{P}_{f,n} = X - Y = 4 - 2 = 2$.

Typical Description of Stage 1

Now, we can typically describe the algorithm for estimating the potential $\mathcal{P}_{f,n}$ of a file f, in a node n after one round. The strategy is performed at the end of each round in three phases, as illustrated in Algorithm 1:

After reading the values from the previous round, D_0 and NR_0 and the value $NR_{f,n} = 83$ from the end of this round (inputs of the algorithm, line 3) it sets T_x and T_y (line 4). Then, using (9)-(13), it computes the demand values for the eight leaf nodes at the last level of the tree (line 5).

The loop in lines 6-10, estimates the number of requests in the leaf nodes, for the given input. For the tree of Fig.3(b), the outer loop (line 6) is executed $2^3 = 8$ times and the inner loop (line 8), three times per node. Assume that, the nodes are indexed from left to right and consider the leftmost node $j = 1$, which is reached via a $y3-$path. For node $j = 1$, line 8 is executed three times giving:
$NR_j = 55 + 5 \times 0.5 = 57.5$, for $k = 1$
$NR_j = 57.5 + 2.5 \times 0.5 = 58.75$, for $k = 2$
$NR_j = 58.75 + 1.25 \times 0.5 = 59.3$, for $k = 3$
Note that the NR values of the tree contain only integers, so the value of node $j = 1$ is written as 59 instead of 59.3. Similarly, for $j = 2$ (the second leftmost node), the path is yyx, so the computations will be:
$NR_j = 55 + 5 \times 0.5 = 57.5$, for $k = 1$
$NR_j = 57.5 + 2.5 \times 0.5 = 58.75$, for $k = 2$
$NR_j = 58.75 + 1.25 \times 2 = 61.25$, for $k = 3$

Each pair of nodes $(j, j+1)$ forms an interval I. Thus, we have a total of $2^r/2 = 2^{r-1}$ intervals (line 14). In our example, we have four intervals $I_1 - I_4$.

Phase 2 uses (8) to estimate the upper node values (number of requests). Finally, Phase 3 estimates the potential. Initially, it reads the number of requests for file f at the end of the current (round) and then it finds all paths in the tree leading

input : Number of slots per round, r
From previous round: D_0, NR_0 for f.
From this round: $NR_{f,n}$, for f.
output: A binary tree with possible scenarios regarding the progress of f and $\mathcal{P}^{f,n}$, the potential of f in node d.

1 **for** *every file* f **do**
2 *Phase 1*;
3 Read D_0 and NR_0 ;
4 Set $T_x = \frac{1}{T_y}$ // (*only for this phase*)
5 Estimate the demand values for the 2^r nodes at the lowest level of the tree using (9)-(13)
6 **for** $j \leftarrow 1$ **to** 2^r **do**
7 $NR_j = NR_0$
8 **for** $k \leftarrow 1$ **to** r **do**
9 **if** *path[k]=*x **then** $NR_j = NR_j + D_{k-1}T_x^k$
10 **else** $NR_j = NR_j + D_{k-1}T_y^k$;
11 // *This estimates the NR node values at the last level of the tree*;
12 **end**
13 **end**
14 Set the intervals formed by the node pairs
15
16 *Phase 2*;
17 *//(Bottom-up Tree formulation)*
18 **for** $k \leftarrow 1$ **to** r **do**
19 **for** *every node pair* **do**
20 Choose (x, y) such that: $p = (1-p) = 0.5$
21 *//(Or choose different probabilities)*
22 Use (8) to estimate $NR_{t-1}^{f,n}$
23 *//(Node value one level up)*
24 **end**
25 **end**
26 Set the intervals formed by the node pairs
27
28 *Phase 3*;
29 Read NR_f
30 Start from root
31 **for** *every path leading to* NR_f **do**
32 Count $x-$steps, X and y-steps, Y
33 **if** *path ends at level* $k < r$ **then**
34 Add $r - k$ in total $y-$steps
35 **end**
36 // *for cases where* $y = 0$ *is not used in estimations*
37 **end**
38 $\mathcal{P}_r^{f,n} = X - Y$
39 *//the potential for file* f *in round* r*, at node* n
40 **end**

Algorithm 1: Potential estimation scheme

to NR_f (line 31). For each such path, it counts the sum of $x-$ and $y-$steps (X and Y respectively, line 32). In case that, the path ends before the last level (lines 33-35), say in level k (i.e., NR_f is only found in the intervals formed at level k of the tree), it adds k decreasing paths, with $y = 0$, indicating that there was no subsequent request for f. The potential $P_r^{f,n}$ for file f at the end of round r, is the difference between X and Y (line 39).

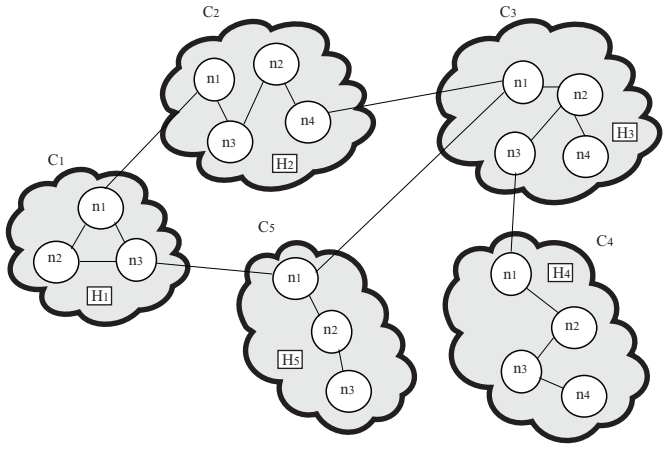

Fig. 4. System framework.

D. Stage 2: Estimating the Scope of a File

Fig. 4 shows a data grid and some connections between the nodes. It consists of a number of clusters, each one having a number of cluster nodes, n and a header node H_i. The header is responsible for managing site information, such as file sizes, distances and shortest paths between nodes, etc. [18] When a node requests a file f, it searches the nearby nodes. If the file is not found, then the request is further forwarded.

By computing the potential of a file, we allow the early detection of popular files which are replicated and become available for future requests. However, there are cases where files are popular only for a short period of time or for a restricted number of users. For example, a project shared by members of a scientific community in cluster C_1 of Fig.4 may have high potential over a number of slots but lower potential after some period of time. The system may become inefficient if we make many replicas of such files.

At the same time, there may be files with lower potential during the first slots, that may be of great interest to a large number of users after some period of time. For example, a news story stored in a node of C_1 may be read only by a small number of users in clusters C_1 and C_4 during the first hours and then become a viral (thus, its potential may increase). In this case, due to geographical locality, it may be requested by many users of C_4, C_3 and C_2. In this case, shouldn't it be preferable to promote for replication the viral instead of the small project?

For such reasons, we introduce the notion of *scope*. By *scope of a file*, we mean the extent to which diverse users are interested or may potentially be interested for a file. If the scope of a file is properly estimated, then the file can be made available to faraway nodes, thus enhancing the system performance. In Fig.4, requests from clusters C_3 or C_4 for a file found in the nodes of C_1 may be long delayed.

To estimate the scope of a file, we first express the length of the path from a requesting node j for file f to node n, in terms of probabilities.

$$\Omega_{n,j} = \left[\prod_{k=1}^{\delta_{n,j}}(1 - prob_k)\right] \times prob_n, \quad (18)$$

where $\delta_{n,j}$ is the distance between n and j measured in number of nodes and $prob_k$ is the probability that a file is cached in a node k. Equation (18) expresses the fact that a request for f is forwarded from j to n through k nodes that do not cache f. This probability is the product of the probabilities that each of these k nodes do not cache f, multiplied by the probability that the $k+1$th node (node n) caches it. For example, if the requesting node is $j = n_1$ of cluster C_3 and $n = n_1$ of cluster C_1, then $\delta_{n,j} = 3$ and we set $prob_k = 50\%$, the probability that f is not cached 1 and 2 nodes away from the requesting node j (that is, in nodes n_3 of C_1 and n_1 of C_5) is $\Omega_{n,j} = \left[\prod_{k=1}^{3}(1 - 0.5)\right] \times 0.5 = 6.25\%$.

Having estimated $\Omega_{n,j}$, the length of the path from a requesting node j for file f to node n in terms of probabilities, we now estimate the scope for file f using the following rules:

1. Requests from a few nodes away are assumed to have zero scope, probably corresponding to files requested for a short time by a small group of users.
2. The lower the value of $\Omega_{n,j}$ for a file f (that is, the larger the distance between j and f), the larger the scope assigned to f.

Based on the following rules, we estimate the scope $S_{f,n}$ of a file f in node n, after one round as follows:

$$S_{f,n} = \sum_{i=1}^{NR} \frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j} \quad (19)$$

where NR is the total number of requests, i indexes the requests for a file f by their arrival time during a slot, j is the node that made request i for file f, $\delta_{n,j}$ is the distance between n and j expressed in number of nodes, $\max(\delta_{n,j})$ is the maximum distance observed in all the requests for f submitted to n by different nodes j and $\frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j}$ is used to assign larger scope values for f when the requests for it are made from faraway nodes. In case that $\frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j} < 0$ its value is set to 0.

For example, consider the following 7 requests for a file f (we further assume that $prob_j = 50\%$) located at node $n = n_1$ in cluster C_1:

i	requesting cluster	requesting node j	distance $\delta_{n,j}$
1	C_1	n_2	1
2	C_4	n_3	8
3	C_4	n_4	9
4	C_4	n_1	6
5	C_4	n_2	7
6	C_2	n_2	3
7	C_2	n_3	2

We have: $\max(\delta_{n,j}) = 9$. Then, from (18), for request $i = 1$ we have $\Omega_{n,j} = 0.5 \times 0.5 = 0.25$ and $\frac{\delta_{n,j}}{\max(\delta_{n,j})} = 1/9 = 0.11$. So $\frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j} \approx -0.14$ and the total contribution of this request to the total estimated scope of f is written as 0, indicating that the request was made from a nearby node. Similarly, for request $i = 4$, $\Omega_{n,j} = 0.5^{10} = 0.000977$ and $\frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j} = 1 - 0.000977 = 99.8\%$. Using (19), we finally find that the estimated scope for f is

$$\frac{0 + 0.884 + 0.998 + 0.651 + 0.769 + 0.208 + 0}{7} \approx 50\%.$$

Using the distance expressed in number of nodes has a huge advantage in computing the scope: this information does not add significant overheads due to the fact that, most routing protocols maintain routing tables that include all information regarding the distance between nodes [22]. Using the routing table, each node can locally find the scope for each file, using just local information.

E. Stage 3: Estimating the Popularity of a File

The total popularity of a file f in node n is computed as follows:

$$FP_{f,n} = \frac{NR_{f,n} \times S_{f,n} \times 2^{\mathcal{P}_{f,n}}}{W} \quad (20)$$

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 and $S_{f,n}$ is its scope. If $\mathcal{P}_{f,n} > 0$, then the number of requests is multiplied by a power of 2, if $\mathcal{P}_{f,n} < 0$, the number of requests is divided by a power of 2 and if $\mathcal{P}_{f,n} = 0$, there is no change. Since the scope is of a file is a percentage, it acts as a controller of the potential of a file. If the file has a high scope, say 90% then its potential will play a major role in the replication policy. If not, then its potential may fall off and the algorithm will assume that the file was only requested locally by a small number of users and, in fact, it has little or no potential.

Generally, more replicas improve data locality. However, if a file has more write than read requests (write intensive), its cost of maintenance becomes higher and efficiency is reduced. To cope with such issues, BTBest uses the write factor W, which is simply the percentage of write requests for a file. This information can be easily obtained locally, by keeping a write counter w and divide w by the number of requests at the end of the round. If W is small, the total potential of a file increases. In case W approaches 1 (a file is used only for write purposes) then, $FP_{n,j}$ approaches $NR_{f,n} \times S_{f,n} 2^{\mathcal{P}_{f,n}}$ and it must have many requests, high scope and high potential to be promoted for replication.

Each node sorts the files in decreasing order with respect to the FP values (i.e., the file with the highest FP values appears first) and selects a percentage of them to be replicated. A system can either use (20) to create one list of the most popular files, or it can create two separate lists, i.e., one that contains the files with the highest NR values and one with the highest $\mathcal{P}$ values and replicate a percentage of files from each list. We will refer to these options in Section 4, where the experimental results will be presented.

Finally, the node sends the information to the cluster header, which decides about the nodes that will accommodate the replicas, as described in the next subsection.

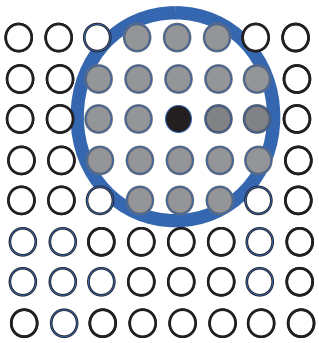

Fig. 5. A simple torus topology. After $R = 3$ rounds, there would be a replica of a popular file in a group of neighboring nodes

F. Stage 4: Computing the Node Suitability

During the last stage of BTBest, the most suitable nodes are selected as targets for each file that has been selected for replication from Stage 2. Assume that n is the node that possesses the file to be replicated and j is a node that needs a replica. There are certain factors we should consider to select the proper nodes:

1. The *number of requests* made in j for f.
2. The *potential* of f
3. The *distance* between the n and j

This information is available to the header file of each cluster after the round. These factors are considered in the following discussion.

During a round, each node receives a number of requests for each file. In cases it possesses the file then these requests are completed, otherwise a copy should be requested. If this file happens to have high potential in every node, then the need for a replica is growing because there is a high probability that the number of requests for this file will further grow. Now, if we set $\mathcal{P}_f$ the potential of f in all nodes after a round, then $\mathcal{P}_f = \sum_{n=1}^{N} \mathcal{P}_{f,n}$, where N is the number of all nodes. Then, an initial approach for the suitability of node j for a file f located at node n will be:

$$SU_{j,f} = w \times NR_j + z \times \mathcal{P}_f \tag{21}$$

where w and z are parameters that assign weights to the two factors. Thus, we have $w + z = 1$. If we want $SU_{j,f}$ to be equally dependent on the number of requests for f in j and on the potential of f then, we set $w = z = 0.5$. Else, if we want one of these factors to be more important, we increase the respective weight parameter and reduce the other one accordingly.

Now, we will try to add one more factor in (21), the distance between n and j, measured by the number of hops between n and j. Apparently, when the distance $\delta_{n,j}$ between the two nodes is small, j becomes a good candidate. We can easily insert this information to the first addend of (21). Particularly, we can write it $\frac{w \times NR_j}{\delta_{n,j}}$. This value becomes smaller as $\delta_{n,j}$ increases, reducing the chances of j to get a replica. However, if there are no requests for f in n, that is, $NR_j = 0$, the distance has no effect on the computation. So, (21) is rewritten as

$$SU_{j,f} = \frac{w \times NR_j}{\delta_{n,j}} + z \times \mathcal{P}_f \tag{22}$$

By choosing the most suitable candidates in such a way, we can assure load balancing in our grid and fast replication process. After R rounds, the nodes that will get a popular file f originating from node n will be at most R hops away from n (if in every round, all replicas are stored in nodes located just one hop away from n). Fig. 5 gives an example of a torus grid, where a file originating from the black node will be stored in each grey node inside the big circle after $R = 3$ rounds (provided that the most suitable candidates are just a hop away from the black node).

G. Space and Time Complexity

Generally, the data replication strategies are methods based on "per-file" information. In this sense, their total cost is dictated by the number of different requested files. For small grids, where the number of requests are generally small, this is not a big problem. However, in larger grids, an increasing number of file requests may cause a dramatic increase in both the storage and time cost, so careful scheduling of the proposed methods is required. Most of the methods (for example [9], [18], [14]) restrict themselves in computing just the file popularity. In [16], the metric of node suitability was also introduced. The BTBest strategy introduces, apart from the above, another two important metrics: the file potential and the file scope. In this subsection, we analyze the space and time complexity and also briefly discuss how the proposed scheme deals with increasing numbers of requested files.

Space Complexity: In Stage 1 of the BTBest scheme, a binary tree needs to be stored in every node and two input values have to be stored: D_0 and NR_0 from the previous round. The binary tree structure is repeatedly used to compute the potential for each file. The potential is then used as input to compute the file popularity. Now, let us assume that F files are requested per node and that two unsigned bytes are enough to store each value involved. Therefore, Stage 1 requires 6 bytes to store the inputs D_0 and NR_0 and the computed potential per file. The size of the binary is *by no means related to the number of files* requested in the node. To compute the potential for each file, we use the same tree structure. An advantage of BTBest is that the intermediate values produced for each file (the values of the tree nodes) need only to be temporarily and not permanently stored. Thus, each node has to store to its memory a maximum of 2^{r+1} values (the number of nodes). Thus, another $2 \times 2^{r+1}$ bytes are required. In total, the storage required for Stage 1 is $6F + 2 \times 2^{r+1} \approx 6F$ since the value of r is relatively small compared to F and the size of the tree structure is added to $6F$ and not multiplied by it.

Stages 2 to 4 of BTBest compute the file popularity, the file scope, and the node suitability. The last stage 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. The other two stages clearly require another $2F$ memory space. Thus, the total memory cost for the BTBest strategy is $6F+2\times2^{r+1}+2F+2F+2F' = 10F+2\times2^{r+1}+2F' \approx 10F$, which represents the cost of storing the two inputs D_0 and NR_0, the file potential, the file scope and the file popularity. For example, if $F = 1000$ and $r = 10$, the BTBest strategy only requires a total storage of approximately 14 KB to compute the proper metrics.

Time Complexity: The total time cost of the BTBest strategy is determined by the cost of its four stages. Stage 1 includes 3 phases. Clearly, Phase 1 requires $2^r \times r$ operations in serial mode. Phase 2 requires $2^r/2 = 2^{r-1}$ computations to estimate the values for level $r-1$ from level r, another $2^{r-1}/2 = 2^{r-2}$ computations to estimate the values for level $r-2$ from level $r-1$ and generally 2^{r-k} computations to estimate the values for level $r-k$ from level $r-k+1$. These computations are 2^r-1 in total. Phase 3 is similar to a tree traversal, which can be performed in r steps. Thus, the total computations per file in Stage 1 are $2^r\times r+2^{r-1}+r$. For a number of F requested files, the total cost of Stage 1 is $F \times \left(2^r \times r + 2^{r-1} + r\right)$. As F increases, it dominates the total cost of Stage 1. To relieve this case, we can take advantage of the binary tree structure and assuming that each node has p processors, divide it into p parts. Then, all parts are processed by different processors in parallel, yielding a total time cost of $F \times \left(2^r \times r + 2^{r-1} + r\right)/p$. Stages 2 and 3 include "per-file" computations, so their costs are also dominated by F, while the time required for Stage 4 (node suitability) is dominated by F', the number of files selected for replication. These stages can also be parallelized, giving a total execution time of F/p. In total, the time required to compute all the metrics required by BTBest is $\left[F \times \left(2^r \times r + 2^{r-1} + r\right) + 2F + F'\right]/p$.

An idea to further reduce this cost is to have each processor pipeline the three expensive stages. Each node requires a 4-segment pipeline arithmetic unit per processor, one segment per stage. Segments 1 and 2 that compute the file potential and scope can operate in parallel, since they require different input data. Once they finish the computations for a file, they pass the results to segment 3 which computes the file popularity, while segments 1 and 2 compute the potential and the scope for another file. Once the files are selected for replication, segment 4 can compute the node suitability for each one of them. By pipelining, we manage to "absorb" the timing of the least expensive operations (like the ones on Stages 2 and 3) and have a total cost that approaches the cost of Stage 1. However, the pipeline design should be carefully designed and delays may need to be introduced. For example, Stage 1 requires more time than Stage 2, so their results are not passed simultaneously to segment 3. Thus, some delay may need to be introduced here. The hardware implementation of BTBest is part of our future work.

IV. Simulation Configuration

To evaluate the BTBest strategy, we can either test it on a real grid or use an event-driven grid simulator. Since it is expensive to establish and maintain a real grid, a simulator is usually preferred.

In our experiments, we compare BTBest with the PFRF [18] and IPFRF [16] strategies, which represent state-of-the-art and well-established strategies for data replication and moreover, they are based on dividing the time in rounds, just like BTBest.

To compare the strategies, the zipf distribution is used to produce file requests. The zipf distribution represents a situation where the probability of making a request for some files is higher compared to the probability of requesting other files. This is particularly useful during the slots of BTBest, where we generally require a constant behavior. However, every 10 rounds the sensitivity factor B drops to a value approaching 0, indicating a change in user behavior. The IPFRF strategy uses mathematical computations to model the changes in user behavior.

Since each file is requested with different probability, we use a partitioning scheme (see also [18]) to facilitate our experiments. Specifically, the files are divided into three classes and the files of each class have different probability of being requested. The first class includes the files with high request numbers and high or low potential or scope, the second includes the files with lower request numbers but high potential and scope and the third includes the remaining files. After one or more rounds, some of the files swap classes. For example, a file initially positioned in the third class, after being requested some times, it may move to the second class, increasing its probability of being selected for replication in the upcoming rounds. A file from the second class that did not have a lot of requests although its potential was high, necessarily moves to the third class. Also, files from the first class which are not requested for some rounds may see their potential dropping off, and after some more rounds they have to move to the third class. If the *sensitivity* factor B changes dramatically, for example if it drops from a value close to 1 to a value close to 0 then, these classes are created from scratch. In all experiments, we choose to replicate 15%-20% of the most popular files.

Since PFRF and IPFRF do not mention anything about the scope of a file, the requests for a file from faraway nodes were treated in the same way as the requests for the same file from nearby nodes. The popularity was measured based on the total number of requests (for PFRF) and based on the number of requests and the file size (for IPFRF). For the BTBest strategy, Eq. (19) was used. The distances between the nodes of the data grid were stored in a tabular form in the memory of each node, as suggested by most routing protocols.

In PFRF [18], the node where each replica is stored is not determined, so, to have a fair comparison, we assumed that, whenever a file must be replicated from a node n, a nearby node in the same cluster is checked and if it does not posses the file, the replica is stored there. If not, then the next node is checked and so on. The IPFRF and the BTBest strategy (see

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Number of files	100-200
File size	100 MB-2 GB
Number of rounds	20
Duration of round	3600 s
Number of slots	6
Number of intervals created	64
Probability of increasing demand, p	50%
Number of clusters	8
Nodes per cluster	20
Jobs per node in a round	20
Files required per job	15
Transmission speed inside clusters	1 Gbps
Transmission speed between clusters	100 Mbps
Sensitivity factor B	Close to 1, but approximates 0 every 10 rounds, in some simulations.
Percentage of replicated files	10-20%

Eq. 22) include computations for node suitability, so these computations were used.

Also, PFRF and IPFRF do not consider different file types. However, there is a difference between the two schemes: in IPFRF, the file popularity is generally reduced for large files (although the authors do not mention what kind of files could they be), while PFRF takes no action at all regarding the file size. In our simulations, we assign a probability (no more than 20%, since the majority of requested files are read only) for a file to be processed by other users. When such files are encountered, BTBest reduces their potential, based on the value of the write counter maintained for each file (see Eq. 20).

Also, some common measures have to be defined: (a) the average job execution time, and (b) the total latency (delay).

The *average job execution time* (AJET) is the total time required to execute all the jobs divided by the number of completed jobs (NoJ) and it is one of the most important metrics to evaluate performance. We can consider the average job execution time as a function of the *average job turnaround time* (AJTT), which is the time elapsed from the time a job requests the files needed to the time it receives these files. Thus,

$$AJET = \frac{AJTT}{NoJ} \tag{23}$$

The communication latency can be computed as follows:

$$L = \frac{FS \times \delta_{n,j}}{TS} \tag{24}$$

where FS is the file size of the replica, $\delta_{n,j}$ is the distance between two nodes measured in number of hops, and TS is the transmission speed inside a cluster. In our simulations, we assumed same TS between all the clusters.

Table II presents the main simulation parameters and their values.

V. Simulation Results and Discussion

In this work we use event-driven simulation to evaluate the performance of BTBest. To facilitate our analysis we used a relatively small number of 200 files. We have chosen to run the simulations for 20 rounds, where a round is 3600 s. Also, we set the number of slots to 6, thus creating 64 intervals. For most of our experiments, these intervals were adequate for our estimations. To create the binary trees, we set $p = 1-p = 50\%$ and slightly changed the values of x, y for each level of the tree. Each cluster is equipped with 20 nodes and there are 8 clusters in total. The transmission speed inside a cluster (for example, from user to router inside a cluster) is assumed to be 1 Gbps and the transmission speed from cluster to cluster is assumed to be 100 Mbps. In the following subsection, we compare the BTBest strategy to PFRF and IPFRF with respect to their hit ratios and average job execution times.

A. Hit Ratio

The hit ratio is the number of requests completed at each round divided by the total number of requests. In this set of experiments, we compare the hit ratio of BTBest with the hit ratio achieved in PFRF and IPFRF. We assumed that each node processes around 20 jobs per round, with each job requiring 15 files. This gives 300 file requests per round. Also, we assumed for the first round (before any replications begin) that the hit ratio for the compared strategies is 50%. The number of files was chosen to be 100.

Both PFRF and IPFRF compute each file's popularity metric based mainly on its requests per round, therefore their hit ratios were found almost similar. Actually, the PFRF scheme has slightly better hit ratio when the file sizes increase (regardless of the file usage, read or write), because IPFRF gives lower popularity to large files. On the contrary, the BTBest strategy reduces the popularity only to write intensive files with relatively low potential or low scope, thus its hit ratio for large files is high.

We run three sets of experiments: In the first set, each node replicates 20% of the files, the ones with the highest popularity. As seen in Fig. 6(a), the hit ratio for BTBest increases quickly, reaching $\approx 75\%$ after one round and as more replications occur after consecutive rounds, it keeps growing, reaching $\approx 99\%$. The main reason for this quick increase of the hit ratio, is because BTBest replicates not only files from the first class (with high numbers of requests) but also a number of files from the the second class (with lower number of request but high potential). A lot of these files presented very high request numbers after one or a few rounds. On the average, PFRF and IPFRF, are only creating replicas of files from the second class after 3 or 4 rounds. This is why their hit ratio converges to the one found in BTBest, but only after a some rounds. In this set, the hit ratio of PFRF and IPFRF reached $\approx 93\%$.

In the second set, we reduced the percentage of replicated files to 10% As seen in Fig. 6(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.

In the third set of experiments, we show clearly the effect of the files *potential* on the hit ratio. We arranged each node to create two separate lists for the 20% of its files to be replicated. The first list included 5% of the files with the highest potential and the second included 15% of the files with the higher

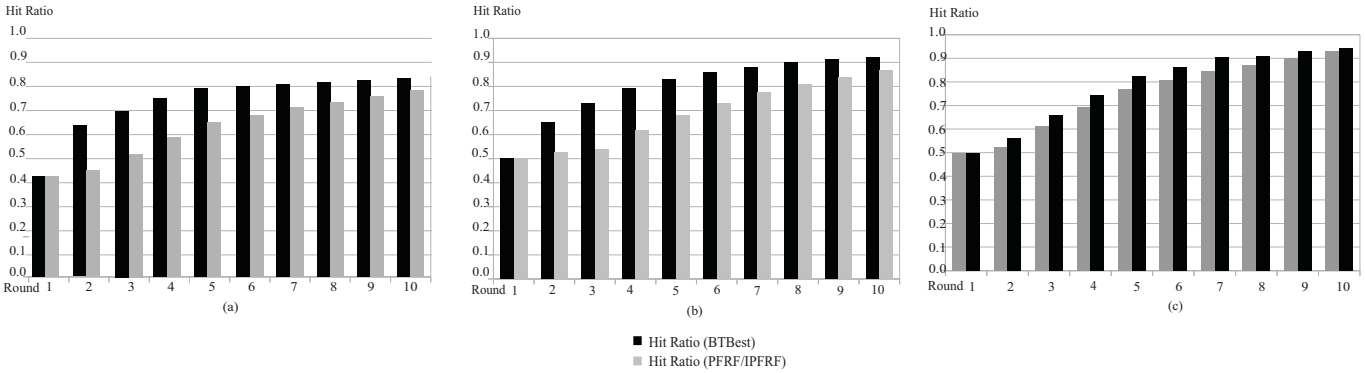

Fig. 6. Comparison of hit ratios between BTBest and PFRF, IPFRF. (a) 20% of the files are replicated, (b) 10% of the files are replicated, (c) 20% of the files are replicated, 15% is separately selected from files with higher NR, 5% is separately selected from files with higher potential

request numbers. The result is that, there is higher convergence of the hit ratio of the compared techniques. By reducing the replicas of files with high potential: (1) The second list almost had the same files after each round, thus a small number of replicas was produced, and (2) A smaller number of files from the second class were replicated per round. As a result, the BTBest hit ratio was reduced and converged to the hit ratio of PFRF and IPFRF, as can be seen in Fig. 6(c).

B. Average Job Execution Time

In this subsection, we compare the AJETs for BTest, PFRF and IPFRF. We conducted two simulation sets to study the effect of file size [23] on AJET: In the first set, the file sizes varied from 100 MB- 1GB, while in the second set, they varied from 1-2 GB. The other parameters were the same as in Table II.

In the first set, the BTBest strategy gives better results during the first rounds especially when there are requests with high potential and scope, since these issues are not handled by the other schemes (see rounds 2-4 of Fig 7(a)). Particularly, because BTBest handles the file scope, it offers better AJETs for jobs running on faraway nodes, in cases the files they require have some potential. After some rounds, PFRF and IPFRF have also replicated some of these files, and due to the temporal locality more requests are completed quickly. Thus, their performance comes closer to BTBest's.

In the second set, we kept the same strategy regarding the replicas, but increased the file sizes from 100 MB-1GB, to 1-2 GB. There are two issues related to the file size: (a) The nodes must have enough disk space or some replacement policy (for example, least recently used) to be able to accommodate the replicas, (b) Larger replicas cause higher communication latency. The communication latency has been defined in (24).

Since the transmission speed inside the clusters is constant, L increases due to FS and this in turn affects the average AJET. To keep L as small as possible, the BTBest strategy prevents the large write intensive files from being replicated by reducing their popularity, as described by (20). Also, it replicates the files of each node n in nearby nodes, as described by (22). As seen in Fig. 7(b), the BTBest technique suffers about 3 times higher AJETs as the file sizes increase, due to increasing latency. The PFRF strategy has no policy regarding the target nodes so its performance dramatically drops as the files sizes increase, because these large files can be copied anywhere, especially if the copies exist in nearby nodes (see the description of the previous section). The IPFRF assigns low popularity values to large files [see [16], Equation (5a)]. Large files are generally not replicated, but this increases the AJET when such files are involved. Its performance is better compared to PFRF, but BTBest overcomes it.

Another thing to note in Figures 7(a,b), is that every 10 rounds (rounds 10 and 20), the users behavior towards the files changes. In this case the AJETs are the same for all files since completely different files are requested. In BTBest, this situation is handled by assigning a value to B [see (7)]. This explains the spikes shown in the Figures.

VI. Conclusions-Future Work

In this paper, we presented BTBest, a binary-tree based strategy that estimates the potential of a file, that is, the increasing or decreasing demand for this file during a period of one round. The main strength of BTBest is that it predicts the increasing demand of high-potential files and makes them available via replication sooner than other well-known strategies. Also, by estimating the scope of the file, it replicates files with some potential that may be widespread in the future. In this way, the hit ratio increases, thus more file requests are completed and as a result, better average job execution times are achieved.

This work has given rise to many issues that need to be further investigated. First of all, the hardware implementation of the BTBest strategy (especially using pipeline-based schemes) is of major importance. Also, BTBest has to be tested in a real data grid. The implementation of BTBest to higher-dimensional trees with the insertion of more options regarding the increasing/decreasing factors is also of great interest. Finally, we plan to adopt BTBest in tree topologies, where nodes can enter and leave the network (due to loss of energy). The scheme can be used to estimate the energy levels that remain in each node based on its traffic. When estimations show that a node may run out of energy, an action should be taken (for example, send some necessary replicas to nearby nodes).

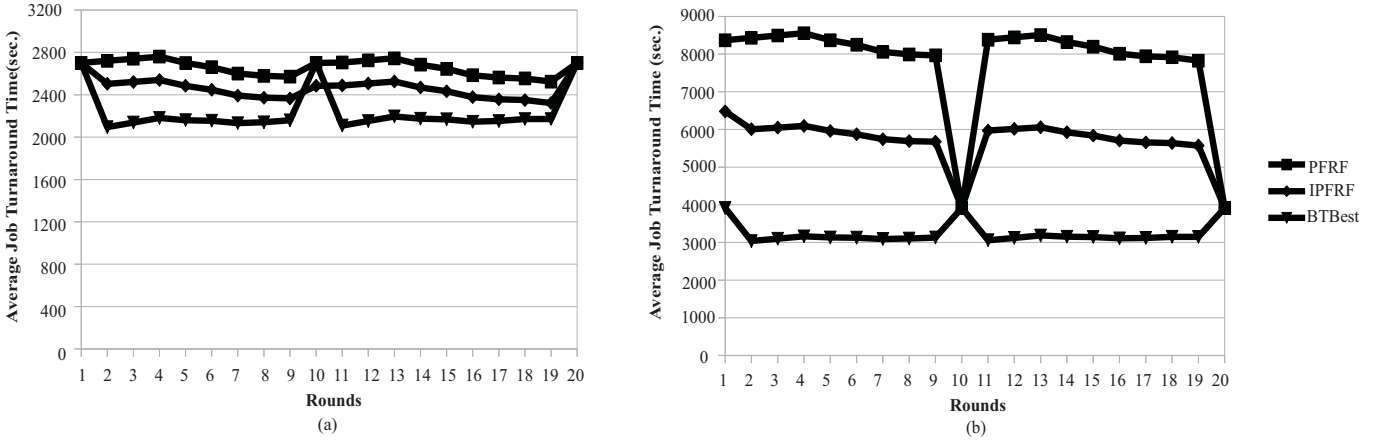

Fig. 7. Comparison of AJETs between BTBest and PFRF, IPFRF (a) File size between 100MB and 1GB, (b) File size between 1 and 2GB

REFERENCES

[1] W. Li, Y. Yang, and D. Yuan, "Ensuring cloud data reliability with minimum replication by proactive replica checking," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1494–1506, 2016.

[2] G. A. Michelon, L. A. P. Lima, J. A. d. Oliveira, A. Calsavara, and G. E. d. Andrade, "A strategy for data replication in mobile ad hoc networks," in *22nd IEEE International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems*, Paris, France, September 9-11 2014, pp. 486–489.

[3] R. Souli-Jbali, M. S. Hidri, and R. B. Ayed, "Dynamic data replication-driven model in data grids," in *39th IEEE Annual Computer Software and Applications Conference*, vol. 3, Taichung, Taiwan, July 2015, pp. 393–397.

[4] E. Spaho, L. Barolli, and F. Xhafa, "Data replication strategies in P2P systems: A survey," in *17th International Conference on Network-Based Information Systems*, Salerno, Italy, September 10-12 2014, pp. 302–309.

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

[6] M. Bsoul, A. Al-Khasawneh, E. E. Abdallah, and Y. Kilani, "Enhanced fast spread replication strategy for data grid," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 575–580, 2011.

[7] 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.

[8] M. Bsoul, A. Alsarhan, A. Otoom, M. Hammad, and A. Al-Khasawneh, "A dynamic replication strategy based on categorization for data grid," *Multiagent and Grid Systems*, vol. 10, no. 2, pp. 109–118, 2014.

[9] 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.

[10] 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.

[11] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2377–2393, 2016.

[12] 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.

[13] M. Shorfuzzaman, P. Graham, and R. Eskicioglu, "Adaptive popularity-driven replica placement in hierarchical data grids," *The Journal of Supercomputing*, vol. 51, no. 3, pp. 374–392, 2010.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] S.-M. Park, J.-H. Kim, Y.-B. Ko, and W.-S. Yoon, "Dynamic data grid replication strategy based on internet hierarchy," in *Lecture Notes in Computer Science*, vol. 3033. Berlin, Heidelberg: Springer, 2004, pp. 838–846.

[20] A. Nahir, A. Orda, and D. Raz, "Replication-based load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 494–507, 2016.

[21] R.-S. Chang, H.-P. Chang, and Y.-T. Wang, "A dynamic weighted data replication strategy in data grids," in *IEEE/ACS International Conference on Computer Systems and Applications*, Doha, Qatar, 2008, pp. 414–421.

[22] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: ACM, 1994, pp. 234–244.

[23] B. Fu and Z. Tari, "A dynamic load distribution strategy for systems under high task variation and heavy traffic," in *ACM symposium on Applied computing*, 2003, pp. 1031–1037.

Stavros Souravlas received the PhD degree in Computer Science from the University of Macedonia, Thessaloniki, Greece. He is currently a lecturer of Computer Architecture and Digital Logic Design in the Department of Applied Informatics, School of Information Sciences, University of Macedonia, where he joined in 2014. His research interests include computer architecture and performance evaluation, parallel and distributed systems, grid computing, cloud computing, systems modeling and simulation. He is the author of 3 books in the fields of Digital Logic Design, Digital Systems Simulation, and Simulation Techniques. He is a member of the IEEE.

Angelo Sifaleras is an Assistant Professor of Mathematical Programming - Network Optimization at the University of Macedonia, Department of Applied Informatics, School of Information Sciences, Greece, Thessaloniki. His research focuses on algorithms and models for network optimization problems.