The International Journal of Parallel, Emergent and Distributed Systems
Vol. 00, No. 00, Month 2011, 1–22

Survey

Trends in data replication strategies: A survey

Stavros Souravlas and Angelo Sifaleras

University of Macedonia, School of Information Science, Department of Applied Informatics, 156 Egnatias Str., Thessaloniki 54636, Greece.

(*v3.6 released March 2011*)

Data Grids allow many organizations and individuals to share their data across long-distance areas. Nowadays, a huge amount of data is produced in all scientific fields and to enhance collaboration and data sharing, it is necessary to make this data available to as many nodes of the grid as possible. Data replication is the technique used to provide this availability. Moreover, it improves access time and reduces the bandwidth used. Recently, data replication has received considerable attention and several new algorithms have been developed. This article provides an overview of the state-of-the-art techniques of data replication. We identify the advantages and disadvantages of these strategies and discuss about their performance.

Keywords: Replication Optimization; Data Grid; File Popularity; Distributed Systems

1. Introduction

Several applications are moving towards a distributed interconnected environment. A Data Grid is such an environment [1–5], where the data storage and all computational resources are distributed throughout different and widespread locations.

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

In such environments, data replication is required so that the users are able to retrieve the requested data from storages residing in nearby nodes [6–12]. Data replication schemes create sets of such nodes, where selected data is replicated, and determines the number of replicas. The overall performance of any distributed environment is crucially affected by the replication strategy it uses.

Data replication arises in several environments, such as the Distributed Database Management Systems. Generally, there is master database environment and one or more client environments. The master environments support read and write operations while the client environments support only read operations. If the master environment fails, applications may assign a client to be the new master. The database environments might be on separate computers, on separate hardware partitions or on separate disks in a single server. In *basic replication* (see Fig.1),

Emails: {sourstav, sifalera}@uom.gr

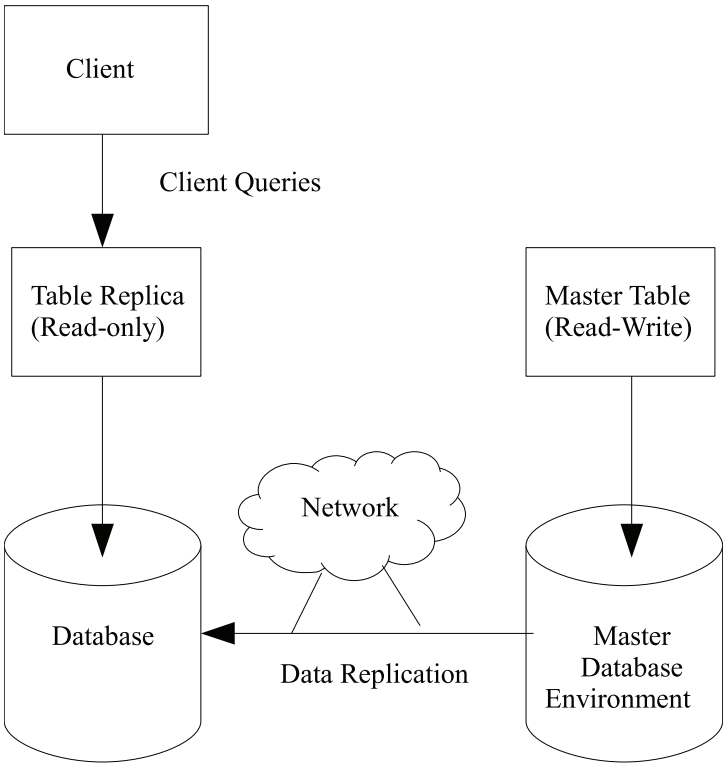

Figure 1. Basic database replication

data replicas provide read-only access to the table data that originates from a primary (master) site. Applications can query data from local data replicas to avoid network access regardless of network availability. The main difference in *advanced replication* is that the applications are allowed to update the table replicas [13].

Another important application of data replication is *disaster recovery*, which refers to programs designed to get a business back up and running after an unexpected interruption of services. A disaster recovery plan designs the response to a disruptive event, to restore crucial business functions. The recovery process includes the restoration or relocation of sources like servers and storage systems, reestablishment of system's functionality, data recovery and synchronization, and finally, restoration of business functions [14].

Many more applications of data replication can be described: Biologists need to share and have available the data about the molecular structure, earth scientists handle and need to have remote access to huge data sets, while data replication in different servers has been used as a means of increasing data availability. Google and Ceph file systems are such examples. Recently, software architectures have been developed, that facilitate the data replication exchange. An example is found in [15].

Data replication can be considered as a system optimization technique that tries to increase the hit ratio, that is, the number of requests completed at each round divided by the total number of requests, reduce the average job execution time (AJET), guarantee high data availability and optimize the use of the network bandwidth.

There are many issues involved in the development of an effective data replication strategy. (1) *Topology:* Each replication strategy has to consider the grid topology, for which it has been designed [16]. Examples include binary tree structures, arbitrary structures where each node can be connected with a number of other nodes without following any rule, torus structures, peer to peer topologies,

etc. Interested readers can find a nice survey focusing on the issue of grid topologies in [17]. (2) *Available space:* The data replication strategies need to consider the available storage space before creating a replica. In several algorithms found in the literature, replacement strategies based on a discipline (FIFO, LRU, etc) are used when there are limitations in the storage available, and (3) *Total Replication Cost:* The total replication cost generally involves the time required to execute the designed replication strategy and the memory this strategy requires. A good scheme should be designed in such a way that, the replication benefits overcome the time and memory losses, (4) *Decisions to be taken:* In all data replication strategies, two important decisions need to be taken: (i) the choice of the proper files for replication, and (ii) the replica placement policy, i.e., the choice of the target nodes to store the replicas and the time that replication will take place.

In this survey, we will classify the data replication schemes based on the last issue, decision making, and discuss about the metrics used to choose the replicas and the placement policies. Our motivation is to present the state-of-the-art data replication strategies from the metric point of view and compare them, so that the researchers can include all the appropriate metrics in their future work. To make the survey more complete, we include a separate paragraph devoted to the comparison between the presented schemes, where the other issues are also discussed. The metrics proposed in the literature consider two important principles: The *temporal locality* and the *spatial locality*. The temporal locality principle states that, the files with high recent demand will probably be requested again soon, with even higher rates. The spatial locality states that, a file related to recently accessed files will probably be requested soon. Also, there is a third principle called the *geographical locality* [18] which states that the files accessed by some users of a grid are likely to be requested by their neighbors as well. This principle formed the basis for a new metric, the file scope, introduced in [19].

The rest of the paper is organized as follows: In Section 2, we review and compare the most important techniques for selecting the proper files for replication. The techniques are divided into time-based (using the principle of temporal locality) and space-based (using the principle of spatial locality), according to the criteria they use (temporal or spatial). The file scope will be discussed in a separate paragraph, since it is based on geographical criteria. In Section 3 the most important techniques for selecting the target nodes for the replicas, are reviewed and compared. Then, Section 4 presents the basic metrics used to evaluate the performance of data replication strategies. Finally, Section 5 presents aspects of future work on data replication.

2. Selection of the proper files for data replication

The strategies proposed in the literature for selecting the proper files for data replication fall into two major groups, depending on the principle they use to support this selection: the *time-based* strategies and the *space-based* strategies. In this section, we present the most representative algorithms of both families. In the end, we refer the geography-based metric of file scope, which is a metric based on the geographical position of the nodes.

2.1 Time-based strategies

Generally, the time-based strategies try to record the behavior of the grid nodes towards each file, to make a decision regarding which files need to be replicated. We can further divide the time-based strategies into two subclasses: the *static* strategies, that assume the users behavior remains unchanged, at least over a considerable amount of time, and the *dynamic* strategies, that assume the user behavior changes at regular intervals. The static strategies study the user behavior over an integral amount of time. On the contrary, the dynamic strategies study the changes in user behavior by dividing the total time into smaller periods, referred to as *time slots* and studying separately the users' behavior within each slot. In this subsection, we will study the most representative static and dynamic time-based strategies.

2.1.1 Static strategies

Ranganathan and Foster [20], in an early approach to static time-based replication strategies, implemented six different replication strategies, namely, *No Replication, Best Client, Cascading Replication, Plain Caching, Caching plus Cascading Replication, and Fast Spread*. The first strategy simply implies that, there is no replication or caching, apparently for comparison purposes. In the *Best Client* strategy, each node maintains a history for each file that it stores, where information like the number of requests for this file as well as the requesting nodes is saved. Each node compares the number of requests for each file to a predefined threshold. If the threshold is exceeded, the node that has created the maximum number of requests receives a replica. Thus, all files for which the threshold is exceeded have one replica to a best client. Thus, the metric used to select a file for replication is:

$$NR(f) = \text{Number of requests for a file} \tag{1}$$

where $\mathcal{T}$ is a predefined threshold.

In the *Cascading Replication* policy, a threshold is also predefined. Here, a level-based hierarchy is used to represent the data flow. When the threshold value is exceeded at the root, a replica is created at the next level, but on the path from the root to the best client. Thus, the new node that stores the replica is an ancestor of the best client. Once the threshold is exceeded at the second level, another replica is created at the lower level, in a similar manner. Therefore, the metric is similar as defined in Equation 1, but it differs to the way the target for each replica is selected. Target selection will be also discussed in the next Section.

In the *Plain Caching* scheme, the requesting node locally stores a copy. This strategy necessitates large disk space and can be effective only for relatively small sized files. The *Caching plus Cascading Replication* strategy combines the *Cascading Replication* and *Plain Caching* strategies. The files are still cached locally by the client nodes, while the server periodically identifies the popular files and creates copies of them in a hierarchical manner. Finally, in the *Fast Spread* scheme, a replica of the popular files (as described by Eq. 1) is saved in each node on the path to a requesting node.

The *Fast-Spread* strategy gave rise to the *Enhanced Fast Spread, EFS* replication strategy, presented by Bsoul et al. in 2011 [21]. *EFS* addresses the following issue which is present in *Fast Spread*: If the storage of one of the nodes on the path from a server to a requesting node is full, a group of existing replicas should be replaced with the new replica. The *Fast Spread* strategy can use one of the many replacement strategies to determine the replicas that need to be replaced, like Least Recently Used (LRU) or Least Frequently Used (LFU). However, the importance

of the replicas is not taken into account: what if the existing replicas are more important than the new ones?

The authors of EFS aimed mainly at addressing the issue of replica replacement (this will be further discussed in Section 4), but they also added some more factors other than the number of requests, to identify the replicas. Such factors were the replica sizes and the frequency specific time interval. These factors have also been later used in other approaches too.

The *EFS* strategy replaces a group of replicas based on the value of two metrics: the group value (GV) and the replica value (RV). Specifically, a group is replaced only if its value is smaller than the value of the requested replica. The GV and RV values are computed as follows:

$$GV = \frac{\sum_{i=1}^{n} NOR_i}{\sum_{i=1}^{n} S_i} + \frac{\sum_{i=1}^{n} NORFSTI_i}{FSTI} + \frac{1}{CT - \frac{\sum_{i=1}^{n} LRT_i}{n}} \tag{2}$$

$$RV = \frac{NOR}{S} + \frac{NORFSTI}{FSTI} + \frac{1}{CT - LRT} \tag{3}$$

where n is the number of replicas in the group, NOR_i is the number of requests for replica i in the group, S_i is the size of replica i in the group, $FSTI$ is the frequency specific time interval, $NORFSTI_i$ is the number of requests of replica i in the group within the $FSTI$, CT is the current time, LRT_i is the last request time of replica i in the group, NOR is the number of requests of a specific replica, S is the size of this replica, $FSTI$ is the frequency specific time interval, $NORFSTI$ is the number of requests of this replica within the $FSTI$, and LRT is the last request time of this replica. The smallest RV and GV among all replicas indicates the least important replicas and groups, while the largest values indicate important ones.

In [22], the authors introduced the FairShare Replication (FSR) that balances the load and storage usage of the servers of the grid. It takes the number of requests and the load on the nodes into consideration, before determining to store the files or not. Specifically, they use a hierarchical topology, where the children of the same node are considered as siblings. Then, the strategy is implemented as follows: When a client requests a file, the request is forwarded to its parent. If the data is found there, it is transferred back to the client, otherwise the request is transferred to the sibling node. If the data is not found in any of the siblings, the request is transferred one level up and the process continues in a similar manner. The decision regarding the files to be replicated is based on the data access frequency. The system maintains a global workload table G. During specified time intervals, G is processed to get a cumulative table (fileId, ClientId, Frequency), where frequency is the number of accesses. To decide about the replicas, the algorithm uses the *average access frequency* defined as follows:

$$F_{avg} = \frac{\sum freq}{n} \tag{4}$$

where n is the number of entries in G.

In [23], the authors use the popularity value to make replication decisions. Although the strategy claims to be dynamic, the term refers to the dynamic weights and historical access records to determine object popularity, but it does not refer to changes in user behavior. Firstly, they find the average popularity of each object

in a time period T_n and then, they compute the average popularity of the whole distributed system within a period T_n. Based on these computations, the strategy decides if an object needs to be replicated. If the popularity of an object is larger than the average popularity then, the object is replicated and the number of its replicas is computed as:

$$Num\_of\_replicas = \frac{Avg\_obj\_popularity\_per\_interval}{Avg\_obj\_popularity\_of\_whole\_distributed\_system}$$

The authors also presented two extensions of their basic strategy called the *Closest Access Greatest Weight, CAGW*: (1) the CAGW_NP (New Placement) that handles the cases where the system does not include only read-only files or the read/write ratio is high, since the CAGW strategy only pays attention to the read cost, and (2) the CAGW_PD (Proactive Deletion), where the algorithm checks and removes bad replicas, in order to achieve balancing between read and write overheads. The first two strategies delete objects only when there is not enough space on the servers and as long as there is space available, the replication continues, resulting in larger costs that overcome the benefits of replication. Actually, the CAGW_PD poses a threshold on the number of replicas by using the read/write ratio as a metric that determines if deletion is needed. A similar approach regarding the replica placement policy is also followed in [24].

The authors of [25] presented a category-based data replication strategy for Data Grids that considers the fact that, the files existing on a node belong to different categories. Each of these categories is given a value that shows its importance for the node. Thus, when a node has not much storage left, it starts storing only the files that belong to the category with the highest value. Thus, the decision regarding the replicas is strictly based on the *importance* of each file.

2.1.2 Discussion on static time based strategies

As the name suggests, the static time-based replication strategies decide about the replicas in a static way, in the beginning of a time period. Once the decision is made, there are no changes in the strategy despite the possible changes in the users behavior. The access patterns used by the static algorithms discussed, follow the principle of temporal locality. In some of them, the simulations performed indicate an improved performance over random access patterns. This paragraph discusses the performance issues of the strategies described, their strengths and weaknesses.

Ranganathan et al. [20] proposed six different replication techniques for three different access patterns: (1) random, (2) temporal locality, and (3) combination of temporal and geographical locality. The authors used simulation experiments to measure the Total Response Time (TRT) and the Bandwidth Consumption (BC). The experimental results showed that the random patterns are the worst-case scenario, in which, there is a small improvement in response time and bandwidth saving when the plain caching strategy is used. When Cascading and Fast Spread (FS) strategies are used, the improvement in total response time and bandwidth consumption is more significant, since Cascading and Fast Spread use the storage space available at the intermediate tiers, while the other strategies only use the space available at the last tier.

For the temporal locality patterns, the performance improvement was much more considerable than that of random patterns. This is expected since the data replication strategy assumes that, the replicas created will be repeatedly requested. Finally, there is further improvement when some geographical locality is combined with temporal locality in some access patterns because it is assumed that, the files

accessed by some users of a grid will be requested by their neighbors as well.

The multi-tier architecture used in this work is a client-server architecture, however, unlike the other methods presented here, the decision regarding the files to be replicated is not taken centrally by a single server. For example, when the Best-Client strategy is used, each node compares the number of requests for each file to a predefined threshold. If the threshold is exceeded then, the node that has created the maximum number of requests receives a replica. However, the file is replicated only if the number of requests exceeds this threshold. This algorithm is categorized as static, but can be refined to dynamic if the threshold changes according to user behavior.

The work of Ranganathan et al. is important, not only because six strategies were suggested, but also because these strategies were used as a basis for the development of more strategies. However, the proposed strategies have some disadvantages as well, the most important of which are: (1) In the best client approach, since the changes in user behavior are not considered, a client that accesses a file for most of the time may stop requesting it, so this client will not be the best after a while, (2) The fast spread strategy requires unlimited storage. In [17], more disadvantages of this work are mentioned.

In [21], the authors proposed the EFS strategy for temporal access patterns. The strategy considers the frequency of requests, the size of each replica, and the time of the last request for each replica. Due to the temporal locality, files with high frequency of requests that have been requested more recently have higher probabilities to be requested again.

The network topology selected is a complete graph. In this architecture, there is a server and many clients. The server node has the highest capacity and tries to locally complete each request. If a request for a file cannot be completed it is transfered to the nearest client through the shortest path. The authors used simulation experiments to measure the total response time and the bandwidth consumption under three scenarios: (1) Random patterns: In that experiment, the performance was measured under the assumption that the probability of requesting any of the replicas is the same, (2) Temporal patterns with 30% probability of a replica belonging to the Most Wanted Group (MWG), and (3) Temporal patterns with 50% probability of a replica to belong to the most wanted group. The simulation results have shown better performance in the third scenario, because several of the replicas stored in each node belong to the MWG and these replicas are requested more frequently than the other replicas, under the temporal locality assumption. When the probability of making a request for members of the MWG is reduced to 30% (Scenario 2), the total performance is decreased and things deteriorate when random patterns are used (Scenario 1). In [25], the authors checked their scheme to a similar topology and examined similar factors.

The main disadvantage in this work is similar to the first disadvantage mentioned in the work of Ranganathan et al.: there is no care in the changes in the user's behavior; thus some objects found in the most wanted group may not be so popular anymore. On the other hand, the strategy has two advantages: (1) It considers the size of replica to make the replication decision, so it makes better usage of the bandwidth available (not so many copies of very large files), and (2) It overcomes the performance of the original Fast Spread Strategy.

In [22], the authors used similar grid architecture and patterns as in [20], for comparison reasons. The proposed Fair-Share replication strategy performs a little worse compared to Fast Spread in terms of the mean response time, but the later causes higher frequency of replicas, which poses heavy loads on the replica servers, thus high overheads. The fair-share replication generates about 1/3 of the replicas

generated by Fast Spread; thus its network communication cost is reduced.

Wang et al. [23] used two different grid architectures, namely, the multi-tier architecture and the random graph architecture. The random graph network topology was a full connected topology. Each file access number is collected over a constant period and aggregated at the end of this. This file access number is an important factor for making a replication decision, thus, the access patterns used follow the temporal locality principle. For evaluation purposes, the authors evaluated the total Data Transfer Cost (DTC). The simulation results indicated that, the CAGW_PD algorithm has lower DTC than other well-known algorithms, because its replica placement policy considers the unit link cost and update overhead. The strategy presents a good cost model, which is used to capture the minimization of the total object transfer cost. Apart from minimizing the total transfer cost, the replication placement and proactive deletion methods achieve a trade-off between read and write cost. The main drawback of this scheme is that, the algorithm does not perform well when the storage capacity increases. The reason is that, the placement strategy derives from the proposed cost model, which is definitely not a well designed algorithm.

Table 1 summarises the topologies, the access patterns, the evaluated parameters, the advantages and disadvantages of the most important static strategies:

Table 1. Summary of the most important static data replication Strategies

Reference	**[20]**	**[21]**	**[25]**	**[22]**	**[23]**
Year	2001	2011	2014	2008	2012
Topology	Multi-tier	Complete graph	Complete graph	Multi-tier	Multi-tier, random graph
Replication Decision	Client	Server	Server	Server	Server
Access Patterns	(1) Random (2) Temporal (3) Temporal & geographical	(1) Random (2) Temporal	(1) Random (2) Temporal	Temporal Temporal	Temporal Temporal
Parameters Evaluated	TRT & BC	TRT & BC	TRT & BC	TRT & BC	DTC
Advantages	Highly influential	(1) Effective bandwidth use (2) Better than FS	(1) Effective bandwidth use	Less replicas Less communication cost	(1) Reduced DTC (2) Trade-offs between R/W cost
Disadvantages	(1) Best client not always best (2) FS needs unlimited storage	MWG not always MWG	Same as [17]	Worse performance than FS	Performance drops when capacity increases

2.1.3 Dynamic strategies

The strategies presented in the previous paragraph assume that, the users behavior towards the files remains unchanged. In this sense, they are considered to be static. On the contrary, the dynamic time-based strategies are the once that take into account the changes in user behavior. This paragraph presents the most important of these strategies. An early approach to dynamic time-based replication strategies was the so-called *Latest Access Largest Weight*-LALW [24]. This strategy divides the total time into time intervals and the user's behavior is studied within these intervals. The LALW strategy is based on keeping historical data access records assigned with different weights. This affects the importance of each record. A more recent data access record is assigned a higher weight to indicate that, this record is more closely related to the current data access needs. The record is stored in the form of $\langle timestamp, FileId, ClusterId \rangle$. This indicates that, the file with id *FileId* has been accessed by a site located in the cluster *ClusterId* at time indicated by *timestamp*. Each site regularly sends its records to the cluster header.

All records in the same cluster will be aggregated and summarised by the cluster header. Based on the information of all access records, the most popular file is computed. To show the importance of a file over a certain period, the computation is performed using different weights for different time intervals. More precisely, if n periods have passed, the earliest is assigned a weight of 2^{1-n}, the next to earliest is assigned a weight of 2^{2-n}, and the most recent is assigned a weight of $2^{n-n} = 1$. Thus, this weight factor is halved between successive periods. This is the big strength of this method of computation, as recent popular files are considered to be better candidates for replication. The drawback is the extensive use of memory, where the access log files are stored. When the number of files becomes large, a lot of storage is wasted for the historical data access records. Table 2 shows an example:

Table 2. Log file example

FileID	*NodeID*	*Period*	*Req_Num*
f_1	1	1	NR_{f_1}
f_1	1	2	NR_{f_1}
f_1	1	3	NR_{f_1}
f_1	1	4	NR_{f_1}
f_2	1	1	NR_{f_2}
f_2	1	2	NR_{f_2}
f_2	1	3	NR_{f_2}
f_2	1	4	NR_{f_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
f_F	1	4	NR_{f_F}

Based on the above description, the metric used to find the most popular file was the *Access Frequency (AF)*. More precisely:

$$AF(f) = \sum_{t=1}^{N_T} (a_t^f \times 2^{-(N_T - t)}), \forall f \in F, \tag{5}$$

where N_T is the number of time intervals passed, F is the set of requested files, and f is the number of times a file f has been accessed during a time interval t.

To the best of our knowledge, the first strategy to divide the total time into rounds in order to study the changes in user behavior, was named PFRF (Popular File Replicate First-IPFRF), proposed by Lee et al. [26]. In PFRF, the popularity of each file is calculated at the end of each round, and only a percentage of the most popular files is replicated. The measure used to decide about the replicas is the *Popularity Weight* $PW_c^r(f_i)$ for a file f_i in cluster c during round r. The popularity weight is computed as follows:

$$PW_c^r(f_i) = \begin{cases} PW_c^{r-1}(f_i) + A_c^r(f_i) \cdot a, & \text{if } \mathrm{A_c^r(f_i)} > 0 \\ PW_c^{r-1} - b, & \text{otherwise} \end{cases} \tag{6}$$

with a, b being constants. If $A_c^r(f_i) > 0$ there have been requests for f_i, so the popularity weight increases by $A_c^r(f_i) \cdot a$, otherwise, it decreases by a constant b.

In [27], the authors proposed an improved version of PFRF, the *Improved Popular File Replicate First-IPFRF*. As in PFRF, the authors divide the total time into time

slots, but their computation of the file popularity includes other factors such as the replica sizes and the frequency specific time interval. Specifically, the popularity of a file i in cluster c during a round n (a round is a time period) is computed as:

$$FP_{i,c,n} = \frac{FP_{i,c,n-1} + (NOR_{i,c,n} \times a)}{FileSize} \times \frac{NOR_{i,c,n}}{TNOR_n}, \tag{7}$$

where $NOR_{i,c,n}$ is the number of requests for a file i in cluster c during a round n, $TNOR_n$ is the total number of requests at round n and a is a constant. The file size plays a key role in the computation of popularity and usually, small files have higher probability to be selected for replication. The authors justified this decision based on the fact that, the storage space is always limited. However, if there was no request for a file during a round then, its size plays no role at all.

In [19], the authors introduced the notion of *file potential* into the computation of file popularity. The potential of each file was computed using a binary tree mechanism. The high-potential files are considered to have high number of requests in the near future; thus they are promoted for replication. In this sense, they become available sooner than they would be, based only on their access numbers. The user behavior is modeled by a single parameter B, which lies in the interval $[0 \dots 1]$. If B approaches 0 then, the users behavior has changed completely and the computations for file potential and popularity are performed from scratch. The total popularity of a file f in node n is computed as follows:

$$FP_{f,n} = \frac{NR_{f,n} \times S_{f,n} \times 2^{\mathcal{P}_{f,n}}}{W} \tag{8}$$

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 (it will be discussed later in this section). If $\mathcal{P}_{f,n} > 0$ then, the number of requests is multiplied by a power of 2, otherwise if $\mathcal{P}_{f,n} < 0$, the number of requests is divided by a power of 2, and finally if $\mathcal{P}_{f,n} = 0$, there is no change.

2.1.4 Discussion on dynamic time-based strategies

The main advantage of the dynamic data replication strategies consists in the fact that, they consider the change in users' behavior. Again, the access patterns used by the static algorithms discussed follow the principle of temporal locality. This paragraph discusses the performance issues of the strategies described, their strengths and weaknesses.

In [24], the authors proposed a dynamic replication algorithm that, uses access weights to keep track of the access history of all files. The data access history is used to replicate data at frequent time intervals. The network topology selected is a multi-tier grid and the data replication decisions are centrally taken. The simulation parameters evaluated are the Job Execution Time (JET), the storage usage, and the Effective Network Usage (ENU), which is the ratio of the sum of the number of accesses of a file from a remote site, and the total number of file replications to the number of times that a file is read either from a remote site or locally. A lower value indicates a more efficient bandwidth use. The simulation results have shown that, the total job execution improves by 15% compared to the least frequently used (LFU) scheme. In addition, the ENU of LALW is lower by 12% compared to that of LFU. The reason is that, the LFU algorithm always replicates so the ENU increases. Because the LFU algorithm always replicates, the

LALW strategy also performs better in terms of storage usage.

The main drawback of that work is that it cannot gather enough information for the data access history, if the time intervals are too short. Also, the information gathered can somehow be unnecessary if the time intervals are too long. On the other hand, LALW is a dynamic strategy that exploits the temporal locality by frequently updating the data access history. Thus, it achieves efficient data replication and its implementation cost is not so large.

In [26], the Popular File Replicate First algorithm (PFRF), was proposed and a star-topology data grid architecture was used. The PFRF periodically calculates file access popularity to keep track of the users' behavior. The access patterns follow the temporal locality. The authors measure two parameters to test the strategy performance: (1) the average job turnaround time (AJTT), which is an average time interval from the time point when a job sends a file request to the time point when the requested files are received by the job, (2) the average bandwidth cost.

The PFRF was compared to other schemes that also provide shorter job turnaround time and high network usage. However, these strategies do not consider the changes in users' behavior; thus the PFRF schemes improves these limitations. Apparently, the simulation results show that PFRF outperforms all the compared algorithms in terms of AJTT and average bandwidth.

The two aforementioned strategies are typical dynamic data replication schemes, that frequently compute the file popularities to track the variation of users' behaviors. However, they have two important drawbacks: (1) They are not round-based. By dividing the time into shorter rounds, the decision process regarding the files to be replicated is enhanced, since the decision is based on larger and larger numbers of requests; thus it is more accurate, (2) They only consider the number of requests to determine the file popularity.

To overcome these issues, the IPFRF strategy divides the time into rounds of fixed length. Then, at the end of each round, it determines the files to be replicated. To compute the file popularity they also take into account the file size [see Equation (7)], which is important due to the limited storage of the cluster nodes. The IPFRF strategy uses a random graph topology and evaluates three parameters: the average files per request, the average file bandwidth consumption per request, and the percentage of files found. Because of the suggested improvements, the simulation results indicate that the the proposed strategy outperforms PFRF. The main drawback of IPFRF is that it does not consider the potential of a file, which indicates the files predicted to have high numbers of requests in the near future. Also, it does not consider the file scope (See Section 2.3).

This issue was dealt with, by the BTBest strategy, proposed in [19]. The file potential is used to model the cases when a non-popular file is selected for replication because it shows that, it has the potential to be highly requested in the near future. Another measure that enhances this strategy is the file scope, that is discussed in Section 2.3. The authors used a random graph architecture and computed two metrics to evaluate the BTBest strategy: (1) The AJET, that is the total time required to execute all the jobs divided by the number of completed, and (2) The hit ratio, which is the number of requests completed at each round divided by the total number of requests. The BTBest outperforms the PFRF and IPFRF strategies. Specifically, the BTBest strategy performs better as the percentage of files to be replicated increases, for relatively small-sized files. The main drawback of BTBest strategy is that its cost depends on the number of files, so some improvement is required to minimize this cost as much as possible. This matter was partially addressed in [28].

Table 3 summarizes the topologies, the access patterns, the evaluated parameters,

the advantages and disadvantages of the most important dynamic strategies:

Table 3. Summary of the most important dynamic data replication strategies

Reference	**[24]**	**[26]**	**[27]**	**[19]**
Year	2008	2012	2016	2017
Topology	multi-tier	star	multi-tier	random graph
Replication Decision	server	server	server	server
Access Patterns	temporal	temporal	temporal	temporal
Parameters Evaluated	(1) JET (2) ENU	(1) AJTT (2) bandwidth cost	(1) avg file per request (2) avg file bandwidth consumption per request (3) % of files found	(1) AJET (2) Hit ratio
Advantages	(1) Exploits temporal locality (2) Efficiency with low cost	(1) Efficiency with low cost	(1) Outperforms PFRF (2) Bandwidth utilization	(1) File potential (2) File scope (3) Outperforms IPFRF
Disadvantages	(1) Data gathering when time intervals are too short/long (2) Not round-based (3) Popularity considers only no. of replicas	See 2, 3 of [20]	(1) Ignores file potential (2) Ignores file scope	(1) Cost depends on num. of files

2.2 Space-based strategies

The vast majority of the recent works on data replication (such as the ones described in the previous paragraph) focus on temporal locality. Spatial locality was usually used to predict future requests by finding the relationship between certain files and the currently accessed file. The space-based strategies usually focus on the identification or, possibly, on the prediction of the relationship between files. A successful identification or prediction of the relationship between files can make the replication procedure more efficient since, the space-based strategies rely heavily on this relationship. For example, a student that requests the code-files of an application, will most likely request the help files for the application produced.

An interesting predictive approach, *Predictive Hierarchical Fast Spread, PHFS* was proposed in [18]. When a file is requested for the first time, a *Predictive Working Set* (PWS) is created for it in the system. This method uses predictive methods based on data mining techniques that, predict the future needs. These techniques include association rules and clustering and uncover knowledge, such as files accessed together or access patterns that appear frequently. A PWS is an arrangement that, shows the relationship between a randomly selected file B to another file, say A. This relationship is determined by a constant a, where $0 < a < 1$. Due to spatial locality, the relationship between files B and A indicates the probability of requesting B immediately after requesting A. Then, the PHFS strategy predicts the successive file requests based on the relationships between the files.

The technique includes a mechanism for keeping the previous file accesses, analysing them and find the relationships. One of the components included in the mechanism is an initial replication component that finds the related files of a file that is requested for the first time and creates the PWS. Another component called *analyzing component* provides the information required to find the files related to this specific file. The time is also divided into intervals. At the end of each interval, a priority is assigned into each PWS that is valid for the next interval. The priority for each PWS is computed as:

$$\text{Priority} = \text{access numbers} \times \text{ access rate} \tag{9}$$

Based on the priorities, the PHFS strategy proceeds to determine how the replication will be configured. The PHFS method uses spatial data patterns and therefore, it is more suitable for applications in which the users work with the same projects or files. This means that the data requests are not random, but somehow related. A multi-tier data grid architecture was used and the replication decisions were centrally made by the server. The authors used a theoretical example to compare their strategy to Fast Spread. The example was presented on the basis of spatial locality which suggests that the users who work on the same context have almost similar, predictable, requests. From the example, they drew the conclusion that, when the requested files present higher spatial locality, the proposed strategy offers improvement in the average access latency. However, the fast spread strategy has mainly used temporal access patterns; thus the suggested improvement seems rather normal.

In [29], the authors proposed a method called *Partitioned Context Modeling* (PCM). In PCM, the idea of *model contexts* as sequences of file system events was introduced. All previously seen contexts are stored and each node stores a file system event. Through its path from the root, each node represents a sequence of file system events or a context, that has been previously seen. Also, the number of times a sequence occurs is kept. To determine future events, the model is updated by using a pointer array, 0 to m, that indicates the nodes representing the current contexts (C_0 to C_m). Any time a new event E occurs, the children of each old C_k are examined, searching for a child that represents E. If such a child exists, then this sequence (the new C_{k+1}) has occurred before, and is represented by this node's child, so we set the new C_{k+1} to point to this child and its count increases by one. If no such child is found then, this sequence occurs for the first time, so a child denoting E is created and the $k+1$th element of the array is assigned to point to its node. Once each context is updated, a new state of the file system is formed.

Having formed the new file system, the authors select the events to be prefetched using the formula $\frac{Count_{Child}}{(Count_{Parent}-1)}$. This formula computes the probability of occurrence of that child's event. This probability is compared to a parametric threshold to determine whether the data corresponding to this event should be pre-cached or not. Similar approaches are followed in [30, 31].

For evaluation, the authors used a trie, a structure based on a tree. The authors performed a series of tests for spatial access patterns that indicate that, predictive prefetching can significantly reduce I/O latencies and the total runtime, as far as the benchmarks used. However, there were two drawbacks: (1) these tests represented a system with many limitations compared to actual computer workloads and (2) The tests repeatedly used exactly the same data patterns.

2.3 Geography-based strategies

A Data Grid system generates large or even huge replicas in geographically distributed locations, so that users can access them to get better performance and quality of service (QoS) [32]. In this subsection, we refer separately to a geographical-based metric, the *file scope*, introduced in [19]. The file scope deals with 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 may have high potential over a number of slots but lower potential after some

period of time. If many replicas of such files are created, the system may become inefficient. On the other hand, 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 may be read only by a small number of users during the first hours and then become a viral (thus, its potential may increase).

By *file scope*, we denote the extent to which diverse users are interested or may potentially be interested for a file. The scope $S_{f,n}$ of a file f in node n, is estimated as follows:

$$S_{f,n} = \sum_{i=1}^{NR} \frac{\delta_{n,j}}{\max(\delta_{n,j})} - \Omega_{n,j} \tag{10}$$

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.

Returning to Equation (8) note that, if a file has a high scope then, its potential will play a major role in the replication policy. Otherwise, its potential may fall off and the algorithm will assume that, the file was only locally requested by a small number of users and, in fact, it has little or no potential.

3. Replica placement policies

The main goals of the replication placement strategies are the following.

1. *Take a replication decision:* Decide whether replication should take place, when a file is not available in a node's storage.
2. *Replica selection:* Select the best of the replicas, if the strategy decides that replication should take place.
3. *File replacement:* Replace some files, in case there is not enough space to store the replica.

Several approaches have been introduced regarding the placement of the selected replicas. The replica placement policies enhance the data replication strategies since if the selection of storage for the replicas is not defined in a strict way or if the selection is random, then: (1) the data files may not be evenly distributed over the grid, for example one or two nodes may be overloaded with replicas, (2) several large files may unnecessarily be copied in more nodes than it is required; thus reducing the system performance. Some of the strategies described in the previous section are equipped with such a placement policy. However, there are some more papers, which focus mainly on the replica placement policy [33–36]. These strategies cannot be categorized based on our taxonomy, but they have been influential in the design of good replica placement policies. This section describes the most important replica placement policies found in the literature.

Ranganathan and Foster developed the Fast Spread strategy in [20]. In this strategy, a replica of the requested file is stored in each node along its path to the requesting node. When one of these nodes lacks storage, some replicas have to be removed. The strategy creates a list of replicas available in the current node. Then, these replicas are deleted one by one, based on the least recently used principle or

on the LFU principle.

In [33], the authors suggest a simple, stepwise strategy, called *Dynamic Hierarchical Replication* (DHR): When a requested replica is not available in a node, the replica is placed in what is called the *Best Site*. The best site is selected using a list that sorts the nodes based on the number of replica accesses. Then, the replica is placed in the first element of this list. If there are multiple best sites then, the selection is random. As a result, a replica is not placed in all the nodes that have requested it; thus decreasing the memory and job execution cost. The DHR also includes a replacement policy, in case the storage space is not enough to place the replica. As in [20], the idea is to create a list of replicas available in the current node and in the other nodes of the cluster. Then, these replicas are deleted one by one based on the least recently used principle, until space is created. A question that arises from DHR is "what will happen if the new replica is less important compared to the replicas that have to be deleted?" The EFS strategy proposed by Bsoul et. al. [21] answers this question.

The authors used two metrics, the *Group Value (GV)* and the *Replica Value (RV)*. In each node, the replica with the smallest RV is the least important. The strategy replaces a group only if its GV is smaller than the RV of the requested replica. Other factors that affect the computations of GV and RV are the frequency of requests, the last time the replica was requested and the size of the replica. The first three factors show the probability of requesting the same replica again, while the file size is important due to limited storage. The replica policy can be described in the following series of steps:

1. A node uses a replica, if it exists in its memory.
2. If it does not exist, the replica is copied to each node node along its path to the requesting node under the following two conditions: (a) The visited node has enough memory, (b) The visited node does not have enough memory but the replica is more important than a group of stored replicas. Then, this group is replaced by the replica.

Another interesting extension to fast spread is the strategy proposed by Khanli et. al. [18]. This method is called *Predictive Hierarchical Fast Spread* (PHFS). In the previous section, we described how the priority of each PWS is computed. The replica management is performed via a *replication configuration change component*. At the end of each time interval, this component checks the priority of each PWS. It keeps two threshold values T_{max} and T_{min}. If the priority of a PWS becomes less than T_{min} then, the configuration component has to choose some members of the PWS for possible replacement. These members are the least recently used ones. The number of these members depends on the difference between T_{min} and the priority of the PWS. On the other hand, if the priority of the PWS becomes higher than T_{max} then, the replication configuration component chooses the members with higher accesses for replication.

The PHFS strategy models the relationship between files using a complete directed graph that is updated using data mining techniques. The files are represented as nodes of the graph and their relationships are represented with weighted edges. Since the graph is directed, the weights not only indicate a relationship, but also the probability of requesting some files, one after another. Thus, some forms of access patterns are also shown by this graph.

Wu et al. [34] focused their attention on the proper placement of the replicas and the choice of the optimal number of replicas, in data grids with tree topology. The root of the tree is defined as the hub of the data grid. Replicas can be placed on any tree node, except the hub. Users can access files from any leaf of the tree

as follows: First, the file is searched locally within a node. If it is found then it is used, otherwise the request goes up the tree. The first replica found on the way to the root is used. If there is no replica then, the hub provides it. The goal of this replica placement strategy is to place the replicas in such a way that, more requests can be satisfied. The strategy also addresses the following issues: (1) the accurate estimation of how often a leaf is used, (2) the proper placement of the replicas, and (3) the load balancing.

Two models were created: The constrained model that places a range limit to each request, i.e., each request can be served within a number of hops towards the root and the unconstrained model that does not have such limitations. Both models aim to solve the following problems:

1. *MinMaxLoad:* Based on estimations regarding the data usage and given the number of replicas k, the goal is to find a set of tree nodes with cardinality k that minimises the workload.
2. *FindR:* Based on estimations regarding the data usage and given the amount of data D that, a replica or the hub can serve, find a set of the tree nodes with minimum cardinality so that, the maximum workload does not exceed D.

In [23], the authors use the popularity value to make replication decisions. Firstly, they find the average popularity of each object in a time period T_n and then, they compute the average popularity of the whole distributed system within a period T_n. Based on these computations, the strategy decides whether an object needs to be replicated or not. If the popularity of an object is larger than the average popularity then, the object is replicated and the number of its replicas is computed:

$$Num\_of\_replicas = \frac{Avg\_obj\_popularity\_per\_interval}{Avg\_obj\_popularity\_of\_whole\_distributed\_system}$$

Afterwards, a decision is made regarding the nodes where the replicas should be placed. The decision constitutes a stepwise procedure described as follows:

1. In the nth interval, if an object k needs replication, a list with all servers requesting k is created.
2. The list is sorted in decreasing order based on the object popularity.
3. The replicas are stored on the top servers of this list.
4. If the server has no storage available then, a LFU policy determines the replicas that will be removed to enlarge storage availability.

The authors also presented two extensions of their basic strategy. The first is denoted as the *Closest Access Greatest Weight, CAGW*: (1) the CAGW_NP (New Placement) that handles the cases where the system does not include only read-only files or the read/write ratio is high, since the CAGW strategy only pays attention to the read cost. The second is the CAGW_PD (Proactive Deletion), where the algorithm checks and removes bad replicas, in order to achieve balancing between read and write overheads. The first two strategies delete objects only in cases where there is not enough space on the servers and as long as there is space available, the replication continues, resulting in larger costs that overcome the benefits of replication. Actually, the CAGW_PD poses a threshold on the number of replicas, by using the read/write ratio as a metric that determines if deletion is needed. A similar approach regarding the replica placement policy is also followed in [24].

Another replacement strategy was introduced in [35]. The authors used a popularity-based form of the least recently used discipline with one constraint

added, to ensure that there will be no replacement for replicas created in the current time interval. Assuming a multi-tier data grid model, the replication is accomplished in two phases: (1) Firstly, the *bottom-up aggregation phase* aggregates access history records for each file to upper tiers, until the root is reached. During the computation, the access counts are summed, for those records whose nodes are siblings and which refer to the same files. The result is then stored in the parent node. (2) Secondly, by using the aggregated information, the replicas are placed from the top to the bottom of the tree. The idea is to traverse the tree with a top-bottom approach as long as the aggregated access count is greater than or equal to a pre-defined threshold that, determines popular files. A replica is placed at a node if the threshold has such a value that, prevents further traversal through the children of the current node.

An important issue is the determination of the initial threshold value and its adjustment during the execution of the algorithm. The initial threshold value is based on the average aggregated access counts of servers located in the lower tier of the grid. Afterwards, the adjustment is made according to the arrival rate of requests. These changes do not apply immediately but only on a time-interval basis, where a time interval is a fraction of the time required to sample access histories of clients. The threshold value is increased or decreased by the difference between the current and previous average aggregated access counts of servers located in the lower tier of the grid.

A quite simple approach for replica placement is followed in PFRF [26]: After computing the popularity, the authors use the following policy:

1. File selection: the set of popular files is sorted according to the average popularity values in decreasing order and the number of files is calculated.
2. File replication: the strategy checks whether each file is stored in a cluster or not. If so, it takes no action, otherwise it checks for available storage. If storage exists, the file is replicated from the nearest node. In other case, the strategy deletes a number of less popular files compared to the current file.

In [23] the decision regarding the placement of replicas is a stepwise procedure described as follows:

1. In the n-th interval, if an object k needs replication, a list with all servers requesting k is created.
2. The list is sorted in decreasing order according to the object popularity.
3. The replicas are stored on the top servers of this list.
4. If the server has no storage available, a LFU policy determines the replicas that will be removed to enlarge storage availability.

The IPFRF [27] constitutes an extension of PFRF, which computes the suitability of each node as a target for a specific replica, based on the relationship:

$$S_{n,i} = A \times \frac{NOR_{n,i}}{HNOR_i} + B \times \frac{FSS_n}{TSS} + C \times \left(1 - \frac{SOD_n}{HSOD}\right) \tag{11}$$

where $S_{n,i}$ is the suitability of cluster node n for file i, $NOR_{n,i}$ is the number of requests from cluster node n for file i, $HNOR_i$ is the node with highest number of requests i, FSS_n is the free storage of n, TSS is the total storage, SOD_n is the sum of distances between n and the other nodes in the cluster, and $HSOD$ is a node with the highest SOD within the cluster. This approach is very interesting, since it provides three constants A, B, and C, where $A + B + C = 1$, and each

one can be used as a weight for three different factors. In order to select the node with the highest number of requests, we can increase the value of A and decrease the values of B and C. If we want to balance the load between different nodes in the cluster, we can assign a higher value to B and decrease the values of A and C. Finally, if we want to select the node with smallest distance to the nodes in the cluster, we can increase C and decrease A and B accordingly.

4. Performance evaluation metrics

In order to evaluate the effectiveness of their strategy, the authors have considered a variety of different metrics. In this section, we briefly describe the most important of them:

Mean Job Execution Time: The mean job execution time is the total time required to execute all the jobs divided by the number of completed jobs and it is one of the most important metrics to evaluate performance. The mean job execution time can be considered as a function of the *average job turnaround time* (AJTT), which is the time elapsed from the time a job requests the files needed until the time it receives these files. Thus:

$$MJET = \frac{AJTT}{Num\_of\_jobs\_completed} \tag{12}$$

This metric has been used in [26, 27, 33, 35, 37] among others.

Bandwidth Consumption: The file replication takes time and consumes a good portion of the network bandwidth. The policies described in Section 3 aim at reducing the number of replicas, whenever this is possible. A good metric that can be used is the *Effective Bandwidth Consumption* (EBC), which can be described by the following equation:

$$EBC = \frac{num\_of\_remote\_accesses + num\_of\_replicas}{num\_of\_local\_accesses} \tag{13}$$

When EBC is small, more files are accessed locally, resulting in lower bandwidth consumption. Most of the strategies presented in this survey, consider the bandwidth consumption in the simulations performed.

Hit Ratio or Number of Completed Requests: The hit ratio is the number of requests completed at each round divided by the total number of requests. Wei et al. [38] explicitly study the hit/miss ratio in their work, while the majority of the papers presented in this survey prefer to consider the number of completed requests.

Number of replicas and percentage of used storage: Some of the strategies described [9, 24, 34, 35, 37] also study the number of replicas created and the percentage of storage used per node or per cluster since, their main goal is to keep the number of replicas and the storage used to the minimum.

The file size: The file size can't be considered in fact as a metric to evaluate the performance of a data replication strategy. However, simulations are conducted to

study the effect of file size [19, 39] on other metrics, such as the AJET or hit ratio. There are two issues related to the file size, when used to evaluate data replication schemes: (a) The nodes must have enough disk space or follow some replacement policy (for example, least recently used) to be able to store the replicas, (b) Larger replicas cause higher communication latency.

5. Conclusions and future trends

In this survey, we classified the data replication strategies for data grids. We presented the most representative strategies for choosing the proper files for replication, separated into three categories: time-based, space-based and geography-based. The time-based strategies were further divided to static and dynamic.

The strategies have been developed for different access patterns (temporal/spatial) and for a variety of different architectures. The main parameters computed to evaluate these strategies were also discussed. Generally, we can conclude that there is no standard architecture or access pattern used in the papers. In most cases, the multi-trier architecture is used, but the random graph is also a common approach.

There is a variety of parameters that are computed to evaluate the performance of data replication schemes. Some of the most important are the total response time, the bandwidth consumption, the data cost, and the effective network use. There is no single strategy that considers all these parameters, so it is not an easy task to have 100% accurate comparisons between them. Also, the majority of works use simulation for evaluation purposes.

As the volume of data operated by modern applications keeps growing, new challenges are going to be posed in the field of data replication. Future trends may include scheduling of algorithms that optimize the memory usage. As mentioned in Section 2, strategies that keep large access log files like [24] are consuming a lot of memory. An idea to resolve this issue would be to schedule an algorithm that *estimates* the access values for every time interval and uses them as input to compute the popularity of each file. In this way, the system may be relieved from keeping excessively large log files especially now that, the data replication algorithms will have to deal with even larger grids and huge numbers of files.

Another future trend may be the design of new network architectures and hierarchies, that aim to minimise the time spent in moving data between nodes in a distributed system. Finally, it is important to implement data replication strategies in the cloud.

References

[1] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman, *Data replication strategies in grid environments*, in *Proc. of the 5th IEEE International Conference on Algorithms and Architectures for Parallel Processing*, Beijing, China, 2002, pp. 378–383.

[2] A. Nahir, A. Orda, and D. Raz, *Replication-based load balancing*, IEEE Transactions on Parallel and Distributed Systems 27 (2016), pp. 494–507.

[3] C.E. Perkins and P. Bhagwat, *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers*, in *Proc. of the ACM Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, Vol. 24, 31 August - 2 September, London, UK, 1994, pp. 234–244.

[4] N. Mansouri, *Network and data location aware approach for simultaneous job*

scheduling and data replication in large-scale data grid environments, Frontiers of Computer Science 8 (2014), pp. 391–408.

[5] K. Bagheri and M. Mohsenzadeh, *E2DR: Energy efficient data replication in data grid*, Journal of Advances in Computer Engineering and Technology 2 (2016), pp. 27–34.

[6] S.M. Park, J.H. Kim, Y.B. Ko, and W.S. Yoon, *Dynamic data grid replication strategy based on internet hierarchy*, in *Grid and Cooperative Computing (GCC 2003)*, Lecture Notes in Computer Science, Vol. 3033, Springer, Berlin, Heidelberg, 2004, pp. 838–846.

[7] S. Figueira and T. Trieu, *Data replication and the storage capacity of data grids*, in *Proc. of the International Conference on High Performance Computing for Computational Science*, 24-27 June, Toulouse, France, 2008, pp. 567–575.

[8] D.G. Cameron, A.P. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini, *Analysis of scheduling and replica optimisation strategies for data grids using optorsim*, Journal of Grid Computing 2 (2004), pp. 57–69.

[9] W. Li, Y. Yang, and D. Yuan, *Ensuring cloud data reliability with minimum replication by proactive replica checking*, IEEE Transactions on Computers 65 (2016), pp. 1494–1506.

[10] G.A. Michelon, L.A. Lima, J.A. Oliveirade , A. Calsavara, and G.E. Andradede , *A strategy for data replication in mobile ad hoc networks*, in *Proc. of the 22nd IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 9-11 September, Paris, France, 2014, pp. 486–489.

[11] R. Souli-Jbali, M.S. Hidri, and R.B. Ayed, *Dynamic data replication-driven model in data grids*, in *Proc. of the 39th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 1-5 July, Taichung, Taiwan, 2015, pp. 393–397.

[12] E. Spaho, L. Barolli, and F. Xhafa, *Data replication strategies in P2P systems: a survey*, in *Proc. of the 17th IEEE International Conference on Network-Based Information Systems (NBiS)*, 10-12 September, Salerno, Italy, 2014, pp. 302–309.

[13] URL `https://docs.oracle.com/cd/E17276_01/html/programmer_reference/rep.html`.

[14] URL `http://www.oracle.com/technetwork/articles/systems-hardware-architecture/o11-080-`

[15] E. Gallicchio, J. Xia, W.F. Flynn, B. Zhang, S. Samlalsingh, A. Mentes, and R.M. Levy, *Asynchronous replica exchange software for grid and heterogeneous computing*, Computer Physics Communications 196 (2015), pp. 236–246.

[16] N. Mansouri, *A threshold-based dynamic data replication and parallel job scheduling strategy to enhance data grid*, Cluster computing 17 (2014), pp. 957–977.

[17] A. Tehmina, S. Muhammad, and D. Ali, *A survey of dynamic replication strategies for improving data availability in data grids*, Future Generation Computer Systems 28 (2011), pp. 337–349.

[18] 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 27 (2011), pp. 233–244.

[19] S. Souravlas and A. Sifaleras, *Binary-tree based estimation of file requests for efficient data replication*, IEEE Transactions on Parallel and Distributed Systems 28 (2017), pp. 1839–1852.

[20] K. Ranganathan and I. Foster, *Design and evaluation of dynamic replication strategies for a high performance data grid*, in *Proc. of the International Con-*

ference on Computing in High Energy and Nuclear Physics, Beijing, China, 2001.

[21] 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 34 (2011), pp. 575–580.

[22] Q. Rasool, J. Li, G.S. Oreku, and E.U. Munir, *Fair-share replication in data grid*, Information Technology Journal 7 (2008), pp. 776–782.

[23] 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 62 (2012), pp. 227–250.

[24] R.S. Chang and H.P. Chang, *A dynamic data replication strategy using access-weights in data grids*, The Journal of Supercomputing 45 (2008), pp. 277–295.

[25] 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 10 (2014), pp. 109–118.

[26] 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 28 (2012), pp. 1045–1057.

[27] M. Bsoul, A.E. Abdallah, K. Almakadmeh, and N. Tahat, *A round-based data replication strategy*, IEEE Transactions on Parallel and Distributed Systems 27 (2016), pp. 31–39.

[28] S. Souravlas and A. Sifaleras, *On minimizing memory and computation overheads for binary-tree based data replication*, in *Proc. of the 22nd IEEE Symposium on Computers and Communications (ISCC)*, 3-6 July, Heraklion, Greece, 2017.

[29] T.M. Kroeger and D.D. Long, *Predicting Future File-System Actions From Prior Events*, in *Proc. of the USENIX Annual Technical Conference*, 22-26 January, San Diego, CA, 1996, pp. 319–328.

[30] R.S. Chang, N.Y. Huang, and J.S. Chang, *A predictive algorithm for replication optimization in data grid*, in *Proc. of the International Computer Symposium (ICS)*, Taiyuan, Taiwan, 2006, pp. 199–204.

[31] T.M. Kroeger and D.D. Long, *Design and Implementation of a Predictive File Prefetching Algorithm.*, in *Proc. of the USENIX Annual Technical Conference*, 25-30 June, Boston, Massachusetts, 2001, pp. 105–118.

[32] N. Mansouri, *QDR: A QoS-aware data replication algorithm for data grids considering security factors*, Cluster Computing 19 (2016), pp. 1071–1087.

[33] N. Mansouri and G.H. Dastghaibyfard, *A dynamic replica management strategy in data grid*, Journal of Network and Computer Applications 35 (2012), pp. 1297–1303.

[34] 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 68 (2008), pp. 1517–1538.

[35] M. Shorfuzzaman, P. Graham, and R. Eskicioglu, *Adaptive popularity-driven replica placement in hierarchical data grids*, The Journal of Supercomputing 51 (2010), pp. 374–392.

[36] A.M. Rahmani, Z. Fadaie, and A.T. Chronopoulos, *Data placement using dewey encoding in a hierarchical data grid*, Journal of Network and Computer Applications 49 (2015), pp. 88–98.

[37] K. Sashi and A.S. Thanamani, *Dynamic replication in a data grid using a modified bhr region based algorithm*, Future Generation Computer Systems 27 (2011), pp. 202–210.

[38] Y. Wei, A. Aslinger, S.H. Son, and J.A. Stankovic, *ORDER: A dynamic repli-*

cation algorithm for periodic transactions in distributed real-time databases, in *Proc. of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 25-27 August, Gothenburg, Sweeden, 2004.

[39] B. Fu and Z. Tari, *A dynamic load distribution strategy for systems under high task variation and heavy traffic*, in *Proc. of the 18th ACM symposium on Applied computing*, 9-12 March, Melbourne, Florida, 2003, pp. 1031–1037.