IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 20, Issue 5, Ver. IV (Sep - Oct 2018), PP 21-31
www.iosrjournals.org

EDPFRS : Enhanced Dynamic Popular File Replication and Scheduling for Data Grid Enviornment

Durga Shankar Baggam[1], Bibhuprasad Sahu[2]
1(Department of Computer Science & Engineering, Gandhi Engineering College, India)
1(Department of Computer Science & Engineering, Gandhi Institute For technology, India)

Abstract: *A data grid is a structural design or cluster of services that enables transmission of a huge amount of geographically distributed data. Hence, it requires massive storage resources to store enormous data files. Data replication provides a solution for efficiently managing the data files in the huge distributed grid environment. It aids in enhancing the data availability and reducing the overall access time of the file. This paper presents an algorithm called Enhanced Dynamic Popular File Replication and Scheduling (EDPFRS) to enhance the data replication and scheduling of a file based on its popularity and the temporal localities in a data grid environment efficiently. For which a connectivity graph is created for a set of dependent files in each grid site to find the frequently accessed file replicas and maintain the consistency of the updated replicas. The appropriate replica file is located using the parameters such as file access rate and request for accessing the file. A Two-fold scheduling policy is applied to both the master node and head node to reduce the overall data access time in the grid. The best file replica is chosen based on the parameters such as network bandwidth, load rate, and computational capacity of the node. The proposed EDPFRS algorithm excels in replicating and scheduling the files in a grid environment.*

Keywords: *Data Grid, Dynamic Replication, Popular File Replication, Scheduling.*

I. Introduction

Grid provides an infrastructure that involves the collaborative usage of the heterogeneous resources in the distributed network used in various applications such as biometric, weather forecasting, etc., that produce a large amount of data [1-3]. Data grids involve a complete dynamic lifespan of the service placement, provision, management, disintegration and distribution of the data [4]. The huge volume of data in the data grid environment creates new issues like accessing, processing and distributing data in the grids. Hence, data management becomes a challenging task due to the large amount of the data with the complex computations in the grid environment. One of the primary issues of the distributed data grid environment is to optimize the data access and reduce the data access cost in the geographically distributed environment [5].

Replication is the process of generating numerous copies of files on the distributed grid sites, to improve the load balancing among the storage devices, high network performance, data availability and accessibility in the distributed grid environment, where the probability of occurrence of failure is high. If one of the data replica crashes, other replicas are made available [6]. Optimization of the data replication is classified into two types: short and long-term optimization [7]. Static replication is the short term optimization. The location of the replica is fixed at the static replication approach. Dynamic replication is the long-term optimization technique, aimed at reducing the average job access time in the data grid. Dynamic replication has more benefits over the static replication as it can easily adapt to the changes in the data grid environment. Due to the dynamic nature of this approach, the file replicas are created/deleted automatically thereby reducing the data transfers and increasing the data availability [8].

The access time depends on the scheduling of a job for executing a file. Hence, the scheduling process is highly significant for optimally assigning the job to the node having replica. If the jobs are not scheduled properly, there is a huge wastage of the computational resources [9]. This resulted in the irregular distribution of the computational resources in which some nodes are either overloaded or under loaded. Thus, effective scheduling reduces the overall file access time through load balancing across the multiple nodes [10]. Scheduling is associated with the load balancing and resource allocation in the distributed grid environment, while allowing uniform distribution of the workload among the available resources [11].

The Integrated and Adaptive File Consistency Maintenance (IAFCM) algorithm achieves high replica consistence maintenance efficiency at a lower cost in a replica node based on the file query rate and update rates [12]. This paper proposes an enhanced dynamic replication and scheduling algorithm of the popular files in the data grid environment by considering the dependency among the files to replicate a popular group of files. To evaluate the strong dependencies among the files, the proposed EDPFRS algorithm considers the number of file accesses and sequence of file accesses. A connectivity graph is created for a set of files in each grid site based

on the file access history, to find the frequently accessed file replicas and maintain the consistency of the updated replicas.

The most popular files are replicated to the requested grid site. It introduces a new solution that provides replication of popular file group, while considering the dependency among the files to replicate a group of popular files with a strong dependency. The proposed algorithm checks the availability status of a particular node within the region and obtains the decision accordingly. While searching the data to be replicated, there is an increase in the data access speed and reduction in the waiting time. It is necessary to search the amount of recent replicas to be created within the system.

This manuscript is systematized in the following order: Section II presents the prevailing algorithms for dynamic data replication in the data grid environment. Theissues in the dynamic replication and preliminaries of the proposed work are addressed in Section III. Section IV explains about the description presenting operation of the EDPFRS Algorithm. Section V shows a comparative analysis of the proposed EDPFRS algorithm and existing dynamic replication algorithms. The concluding interpretations, advantages and future scope of the proposed work are discussed in Section VI.

II. Related Works

Khanli et al. [4] presented a fast spread algorithm to predict the future requirements and pre-replicate them in a hierarchal way to increase the locality in the file access rate. The fast spread algorithm yielded lower access latency and better performance than the existing common fast spread algorithm. Sashi and Thanamani [7] proposed a modified hierarchical algorithm for dynamic data replication in the data grid system. The availability of data is increased by replicating the files to the region header and storing the replicated files in the frequently accessed site. The proposed algorithm reduced unnecessary replication and job execution time, while ensuring efficient usage of the network bandwidth and storage space.

Saadat and Rahmani [13] formulated a dynamic replication algorithm based on the pre-fetching concept for data replication in data grid. The future requirements of the grid sites are predicted and a series of files is pre-fetched to the requester site, so that the file will be locally available if the grid site is in need of the file the next time. This will significantly reduce the data access latency, response time and bandwidth consumption. However, this dynamic replication algorithm lacks scalability and fault tolerance issues. Mansouri and Dastghaibyfard [14] developed a dynamic hierarchical replication algorithm for placing the replicas in the optimal sites having large number of file access rate. The proposed algorithm prevented redundant creation of file replica that leads to efficient storage utilization.

Tos et al. [15] studied and categorized the dynamic data replication strategies by considering the target data grid architecture. The key points of the strategies are discussed and feature comparison of these key points according to the important metrics is provided. Cui and Zhang [16] introduced a dynamic grid replication algorithm based on the popularity support and confidence for placing the data and its associated replica on an appropriate site, with minimum access latency. The proposed dynamic replication algorithm achieved better performance than the existing replication algorithms.

Mansouri [17] proposed a novel strategy for selecting the replica depending on the response time and security in the distributed data grid environment. Hamrouni et al. [18] focused on the extraction of knowledge to improve the replication of data in the data grids. Maintaining the consistency of the data replica across the distributed data grid environment is a challenging task. Mansouri [19] proposed a dynamic replication and hierarchical job scheduling approach based on the threshold value to improve the data access efficiency in a data grid environment. Abawajy and Deris [20] developed a new data replication protocol to reduce the data update cost and ensure high data availability and consistency in the grid environment. The proposed protocol incurred lower communication and data replication costs. But, this protocol lacks in the security aspect. When a replica updation is required, a distributed system should ensure updating of all replicas. Storing the replicas of same file at different grid sites consumes more storage space. Parallel file management and recovery techniques are also highly expensive. Hence, there is a need for better replication algorithm. In this work, the data availability is increased by placing one replica in each sub-region of the grid. This also aids in reducing the number of replicas within a region and storage cost.

III. Preliminaries

Dynamic replication aims to improve the network bandwidth, data availability and reduce the total access time by considering the following issues

Replica creation

Data replication involves decisions such as duration for creating replica and required number of replica copies. In this proposed work, the replica is created if the file is not available on the node for scheduling the request.

Replica placement

After creating a replica, it is required to decide the placement of replicas for the quicker data access with minimum latency. The replica is placed based on the popularity of the file and available storage space on the node. The file popularity is determined based on the access frequency of the file.

Replica selection

After placing the replica, the best replica is to be selected among the group of available replicas. The criteria for selecting the best replica are based on the availability status, available bandwidth, workload and computing capacity of the node.

Storage space

The amount of storage space should be considered before placing the replica. If the available storage space is less, there is a need for applying the replacement strategies. In this work, the storage space is computed by subtracting the amount of used storage space from the total amount of storage space. The least frequently accessed file is replaced by the replica of the requested file.

Adaptability

The data replication strategy should cope with the dynamic nature of the data grid environment to offer better data access results. If the file is not available during the execution of job, the replica is generated to adapt with the dynamic nature.

IV. Description Of The Edpfrs Algorithm

The EDPFRS algorithm utilizes the following assumptions

- Bandwidth of the nodes located within a sub-region is same.
- Bandwidth between two sub-regions is marginally less than the bandwidth between the nodes lying within the sub-region.
- Bandwidth between two different regions is found to be the lowest amongst the whole bandwidths chosen in the distributed grid environment.
- There can be a single replica relevant to the file in the sub-regions. This replica is placed on the node based on the file popularity.
- As the jobs in a grid site belong to same virtual organization, they have similar interests in the files.
- Jobs have the temporal locality of the file access. The files which are requested currently are having high probability and are to be requested soon.

Fig.1 shows the pictorial representation of grid environment including region, sub-region, master node, and head node [5]. First three assumptions cause the hierarchy of the bandwidth in the data grid. The bandwidth of the intra-subregion is extensive than the inter-subregion that effectively reduces the data access time. Final assumption indicates that the number of file replicas is linearly proportional to the number of the sub-regions, i.e., the number of replicas should not be more than the number of sub-regions lying within a region.

The proposed algorithm is classified into two sections such as region optimizer and sub-region optimizer. Region optimizer is implemented at the master node. The master node comprises an agent that facilitates efficient scheduling of the job to the region depending on the load rate of the head node and bandwidth value between the master node and head node. This aids in obtaining a comprehensive view of all regions in the grid. The sub-region optimizer is referred by the master node based on the information collected by the agent at the master node to obtain the best replica. The sub-region optimizer at the head node comprises the local view of all sub-regions in that region. The local information of a head node includes the availability status of the node, access frequency, computational capability and number of stored files at a node in the sub-region, for scheduling the job to the node through the efficient usage of the resources. Multiple jobs can be executed on each grid site. The number of file accesses should be larger than a threshold weight. There is a great likelihood of dependency between the files when a job on a grid site requests a file and requests another file, when the sequence of file access repeats multiple times.

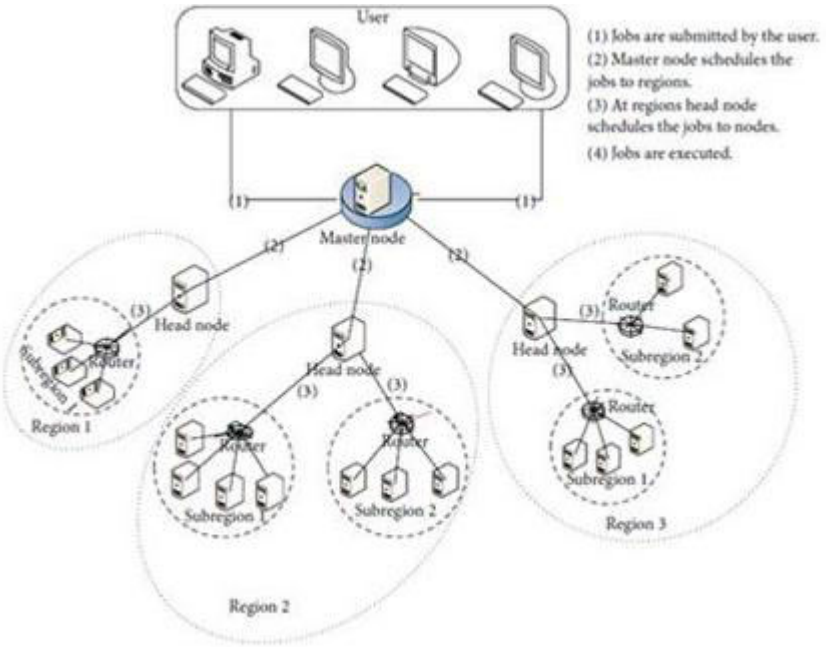

Figure.1 Pictorial representation of grid environment including region, sub-region, master and head nodes

Threshold weight is a predefined threshold value for estimating the dependency between the files. The proposed algorithm replicates all dependent files as a group of files to a requester grid site where the group is accessed frequently based on the assumption that this file group is required in the future. Hence, if a job requests one of the files subsequently, it will be locally available. This reduces the overall job execution time, latency and bandwidth consumption.

The dependency between the required files of each job should be realized to locally replicate those files before receiving the file request. If two succeeding job requests for two different files are repeated, it is logically concluded that there is high dependency between these files. Dependency between two different files is higher, if the number of repetitions of two succeeding requests is increased within a short period. By finding the dependencies between the files and replicating a group of files with a strong dependency is better than replicating the frequently accessed popular files without considering the relationship between the files.

Operation of the EDPFRS algorithm

The algorithm involves three phases for each grid site in a specific time interval and a connectivity graph constructing for a group of files, to find the most popular group of files and placement of replica. In the first phase, dependency between all files are computed based on the file access log of the jobs and file access sequence. These files are stored in the database. In the second phase, the most popular group files are found out for each grid site. In the final phase, the group of files onto the grid site is reunited using the replication process, if a common group of files are frequently accessed by a group of local jobs and a strong dependency exists among a group of local jobs and files.

The proposed algorithm creates a connectivity graph of a group of files for each job on a grid site. A connectivity graph is a weighted undirected graph that represents the dependency of the files toward each other. Each vertex of the graph represents for a file and an edge between the vertices represents the related file accesses. When a job requests a file, the weight of the associated vertex is incremented. This indicates the number of times that the file has been accessed by the job.

If a job requests a file and requests for another file within a specific threshold time, an edge with the weight value of first one is drawn from the first vertex to the second one. If the second file is accessed after the first one within the threshold time and an edge to these files already exists, then the edge weight only is incremented. Thus, each edge represents the number of an individual file accessed immediately after another file. Strong dependency between two different files indicates that when a job requests a file and after some time

the same job requests a new file. The edges between them are not deleted, when the weight of the edges is larger than a certain threshold weight.

Fig.2 shows the connectivity graph of the job executing on the grid site 'g'. To find the accurate dependency between the files, all file access sequences of every job on a grid site are considered as multiple jobs executing on a grid site may request to access the files. Thus, the file requests must be separated from others. Finally, the most Popular Group Files (PGFs) are to be replicated to the distributed grid site environment according to the connectivity graph. When a job requests a file within the time interval if the file had been replicated before the file is available locally in that site. Otherwise, the file would be accessible remotely.

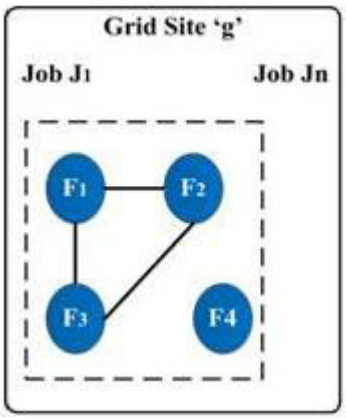

Figure.2 Connectivity graph of a job J_1 running on a grid site 'g'

On every file request, if not an existing one then update the database of a head node with number of file access and the time of last referenced/requested. If the request is an existing one, then the file access rate is incremented or updated. A multi-dimensional matrix named as a connectivity matrix is defined for each job. This matrix is used to store all the accessible files and dependency between the files.

The time difference between two file requests is lesser than the predefined threshold time, so that and are two succeeding files. Then, the weight of the edge between two succeeding files in the connectivity graph is incremented by one. If the time difference between two continuous file requests exceeds the ThresholdTime, this time difference cannot be considered as two succeeding file requests.

A connectivity matrix denotes the number of files accessed by all jobs in that site as the weight of the vertices and the number of files being accessed quickly after each individual file as the weight of the edges between the vertices. Then, graph partitioning is performed to delete the number of edges whose weights are smaller than the Threshold Weight. The undirected graph is divided into the isolated connected sub-graphs including all edges with Threshold Weight. Then, a list of all sub-graphs is arranged according to the total average value of the files accessed in the descending order.

When the user jobs on a distributed environment, the files that are not saved locally are replicated at the end of time interval. The PGFs at the top of the list are replicated to the requester grid site by checking whether the total storage capacity of the site is greater than or equal to the size of the PGF. If the storage capacity is not enough, the grid site should access them remotely. If the storage capacity is greater than the overall size of all PGFs, the files can be quickly replicated to the grid site. Else, some existing replica group in the storage element should be deleted to store the first top PGFs.

Existing PGFs with lower replication cost than the top of PGFs in the list are selected and these selected less frequently accessed PGFs are deleted until the storage space is available for storing the new PGFs [21].

Two-Fold Scheduling

In the proposed algorithm, the job execution request is submitted arbitrarily to the master node. An agent located at the master node performs scheduling of the job to the regions to improve the overall throughput of the data grid. The decision for selecting the best replica and uniform distribution of load on the nodes are based on the scheduling parameters such as network bandwidth, load gauge, and computational capacity of the node. The scheduling helps in reducing the time required for accessing the data in the data grid environment. The available bandwidth between the master and head nodes and the load rate of the head node is checked. This facilitates the master node to balance the load on the regions.

The head node is responsible for scheduling the jobs to the nodes where the replica is placed. An agent is placed on each head node of the region to maintain the information such as node availability, file access

frequency, computational capacity of the node located in the sub-regions. The replica is placed on the nodes based on the high data access frequency and adequate storage capacity. The high access frequency shows the file popularity based on the access log stored at the head node. The file popularity can be decided by the number of requests made for the file while the jobs are executed on the nodes within the region. The available storage capacity 3_8 on a node is based on the storage usage of the node.

Replica Selection

During replica selection, the agent checks the present available status of the node, computational capacity of the node, maximum available bandwidth and load rate. The computational capacity decides the strategy for rapid processing of the job by the node.

After scheduling the request to the neighboring sub-region, the job is processed on the neighboring node. The information of replica in the RC at the head node is updated by the agent of head node. After processing the job, the replica is created in the sub-region to which the job is scheduled primarily. The RC is updated and the job is executed. By increasing the number of replicas within a region, the availability of the file is increased. This helps in reducing the data access time. Hence, there is a tradeoff between the storage capacity and data access time [5].

Replica Management

The replicated files should be maintained within the best sites where the files can be most likely accessed. If the replica does not exist, the least frequently accessed file is deleted and new replica is stored [22]. The original data is replicated automatically when added or modified by the users. When the master replica is replicated completely, each grid site will check the access frequency of replica and storage capacity. The consistency service is executed on each storage element. When the replica updates are transmitted, it is notified to the consistency service. The consistency service should know in advance about the details of the file to be updated to create a temporary copy first. After acquiring a file write command, the current file is updated. When the update process on the file is succeeded, the consistency service is triggered to propagate the changes to the remote grid sites. Furthermore, the RC is updated with the new file modification time [23]. The pseudo code for the proposed EDPFRS algorithm is given below

EDPFRS Algorithm
Submission of job to the master node of grid by the agents/user
Find maximum available bandwidth and minimum load rate
Insert new file
Increment the number of file access
Update the Connectivity Matrix of grid site
if (Gridsite.ConnectivityMatrix(i, j) ≤ThresholdWeight)
Divide the Connectivity Matrix into isolated connected sub-graphs; Obtaining total average of files accesses for each sub-graph;
Arranging all sub-graphs according to their total average of files accesses in descending order;
if (g.StorageSize < Sum(PGF.size))
Access the PGFs remotely;
else
if (g.availableStorageSize < Sum(PGF.size))
Replicate PGFs to the grid site ‘g’ and exit;
else
Calculate Replication Cost for all PGFs
Sort all PGFs in ascending order based on Replication Cost; while (‘selected PGFs’ != empty)
Select files from top of the ‘selected PGFs’ and delete it from grid site ‘g’; Replicate PGFs to grid site ‘g’ and exit; end while
Check existence of file and process the job
Update the file and RC and transmit the replica updates Notify to the consistency service else
if (requested file is not available in local sub-region)
obtain from nearby sub region;
proceed to execute the job with replica
if (free space available in SE of the node where request is initially scheduled)
store new replica
else

check free space in SE of nearby node within sub-region and replicate the file if (there is no enough free space with the sub region) sort the file in least frequently accessed order
if (access frequency of new replica > access frequency of the old file)
delete old file;
if there is enough free space
store new replica;
Schedule job to the region;
execute all the jobs;
end

V. Results And Discussion

The intention of this section is to present the performance analysis result of the proposed EDPFRS algorithm including the test bed, performance metrics and simulation results. The proposed EDPFRS algorithm is compared with the existing method such as BHR, No Replication, Least Recently Used (LRU) and Efficient Dynamic Replication Algorithm (EDRA) [5].

Test Bed

To study the efficiency of the proposed EDPFRS algorithm, a test bed is organized with the following configurations of having two master nodes with the storage capacity of 200 GB and all other nodes with 50 GB, an allowed file size of 1 GB per file and 500 as the maximum job size of 10 different types. The selection of jobs are performed on the combined approach of randomness with their weighted probabilities. The parameters used in this simulation study are listed in Table 1.

T**able** 1 Simulation Parameters

Parameter	Values
Number of grid sites	20
Number of storage elements	20
Number of computing elements	18
Storage capacity of master node	200 GB
Storage capacity of other nodes	50 GB
Types of jobs	10
Number of jobs	500
Number of files accessed by each job	100
Number of simulations	10
Job inter-arrival time (second)	2.5
Size of each file (GB)	1
Total size of files (GB)	350
Scheduling algorithm	Random
Number of repetitions	10 times
ThresholdTime (ms)	40000
ThresholdWeight	35
Time interval	200000 ms

Performance Metrics

The performance of the EDPFRS algorithm is evaluated using the metrics such as mean job execution time, effective network usage, storage usage and total number of replications. In the no replication strategy, the complete data is available at the origin of hierarchy level. In LRU, the file is replicated when it is required. In BHR, a single replica only exists in a region based on the popularity of the file. In the EDRA, the number of file replicas depends on the number of sub-regions.

5.2.1. Mean Job Execution Time (MJET)

MJET is computed as the overall time required to execute a job and the waiting time of a job in the queue to the total number of jobs processed by the system. It is defined as

$$G'' = \frac{\sum_{4}^{I} -'' + H}{6} \quad -4$$

Where 'N' represents the total number of jobs, " denotes the time to execute jobs and H shows the waiting time of the job in the queue.

5.2.2. Effective Network Usage (ENU)

The replication process requires network bandwidth to access a file. The network usage to access a file is computed as

$$G6K = \frac{-6L/ + 6/L + 6M/}{-6NL/ + 6N/L} \quad -5$$

Where $6_{L/}$ represents the number of times the computing element reads file from the storage element on different regions multiplied by the file size, $6_{/L}$ shows the total number of file replications occur during the execution of the job multiplied by the file size, $6_{M/}$ indicates the number of times the computing element reads the file from the storage element on the same sub-region or region multiplied by the file size. $6_{NL/}$ is the time taken to access the remote file and $6_{N/L}$ is the time taken to replicate the file.

Storage Usage

The percentage of storage usage by the files are defined by the storage used with the region under this strategy. It is calculated as

$$9 \quad P\,Q\$ \quad P = \frac{R9_{NCS} - \sum_{4}^{I} 3_{8}T}{9NCS} * 100 \quad -6$$

Where 9_{NCS} represents the total storage capacity of a region and 3_{8} indicates the available storage capacity on a node.

Simulation Results

The proposed EDPFRS algorithm is tested along with EDRA, LRU, BHR and No Replication using the number of jobs ranging from 100-500. Table 2 illustrates the job execution time analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy. With the increase in the number of jobs, the proposed EDPFRS algorithm can process the job within minimum time as shown in Fig.3. The proposed algorithm can access the file within less time and reduce the waiting time, while increasing the file availability. The scheduling strategy can quickly process the workload by replicating the files based on the dependency between the files and scheduling the job to reduce the overall data access time. By finding the dependency between the files and replicating the most popular group of files with high dependency, the required files are available locally during the execution of job.

Thus, the job execution time is reduced as the jobs need not access the required file remotely. When there is a large file overlap between the jobs, all files are shared among all jobs, the MJET of the EDPFRS is minimum. Due to the existence of high temporal locality between the jobs, if the PGFs are replicated, these files will be locally available for the next time. The EDRA algorithm performs better than the BHR, LRU and No Replication. The No Replication strategy performs too worse in all the cases.

Table 2 Mean job execution time of the EDPFRS algorithm and existing EDRA, BHR, LRU and No Replication strategy

Number of jobs	Mean Job Execution Time (ms)				
	LRU	EDRA	No Replication	BHR	EDPFRS
100	44000	40000	90000	42000	38000
200	60000	44000	93000	58000	40000
300	63000	50000	99000	50000	45000
400	70000	61000	102000	76000	57000
500	82000	69000	130000	81000	65000

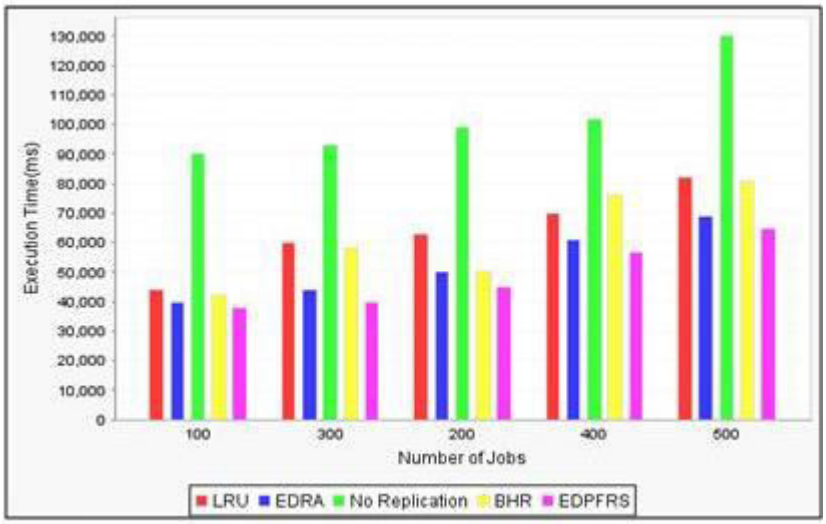

Figure.3 Job execution time analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy

Table 3 Effective network usage of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy

Number of jobs	Effective Network Usage				
	LRU	EDRA	No Replication	BHR	EDPFRS
100	48000	45000	61000	50000	43000
200	60000	46000	80000	70000	42000
300	75000	80000	170000	80000	70000
400	80000	90000	230000	90000	85000
500	100000	95000	310000	110000	92000

Table 3 depicts the ENU analysis of the proposed EDPFRS algorithm and existing EDRA, BHR, LRU and No Replication strategy. Fig.4 shows the ENU analysis of the proposed EDPFRS algorithm and existing EDRA, BHR, LRU and No Replication strategy. The availability of the file increases at the local level by increasing the number of replicas. This reduces the overall file transfer time that results in the less network usage while accessing the file. The proposed EDPFRS algorithm consumes minimum network bandwidth than the EDRA, BHR, LRU and No Replication strategy by avoiding the inappropriate replication of the file and scheduling of the jobs.

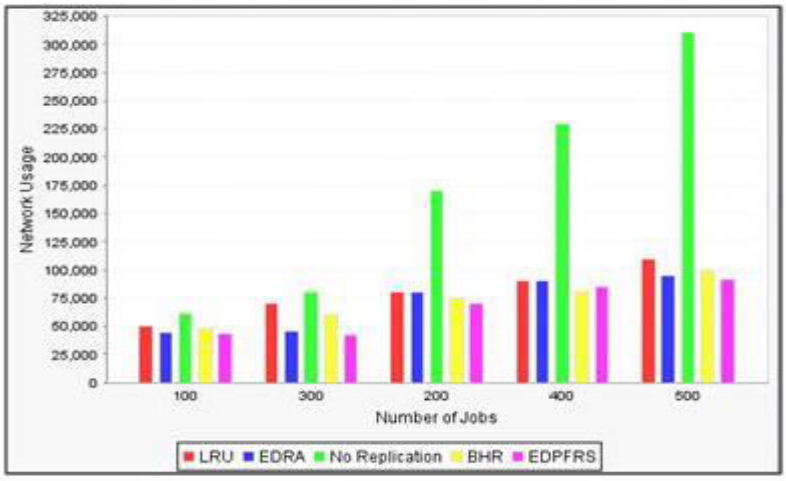

Figure.4 Effective network usage analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy

Table 4 Storage usage analysis of the proposed EDPFRS algorithm and existing EDRA, BHR, LRU and No Replication strategy

Number of jobs	Percentage of Storage usage				
	LRU	No Replication	BHR	EDRA	EDPFRS
100	30	24	33	40	42
200	42	39	42	51	55
300	45	40	53	60	63
400	50	44	55	63	67
500	57	54	63	72	75

Table 4 illustrates the storage usage analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy. Fig.5 shows the storage usage analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy. The No replication strategy requires minimum storage usage as there is no replicas. Due to the increase in the number of replicas, the storage usage of the proposed EDPFRS algorithm is higher than the EDRA, BHR, LRU and No Replication strategy.

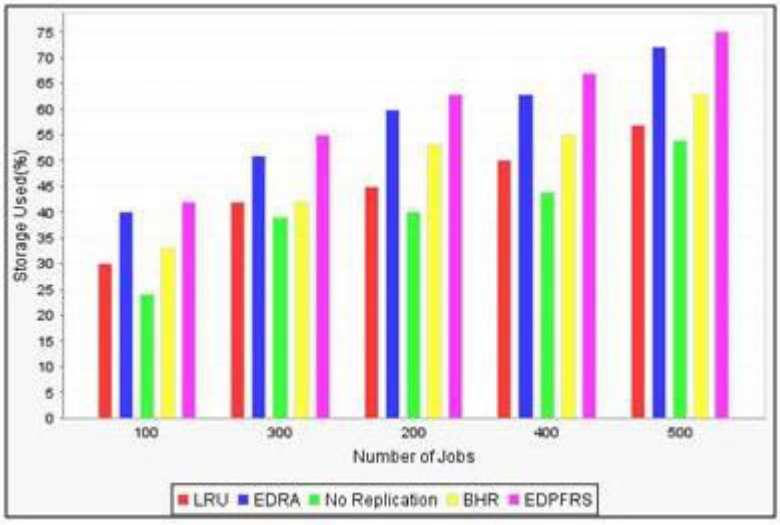

Figure.5 Storage usage analysis of the proposed EDPFRS and existing EDRA, BHR, LRU and No Replication strategy

VI. Conclusion

Data replication is an important phenomenon in the data grid environment, to increase the data availability while reducing the overall data access time and network usage. Most of the existing data replication algorithms have focused on the replication of a single file or a group of files without considering the dependencies between the files. The proposed EDPFRS algorithm considered the dependency between the files and replicates the most dependent files to the requester grid site. The application of two-fold scheduling policy at the master and head nodes help to improve the performance of the proposed EDPFRS algorithm. The EDPFRS algorithm increases the availability of data by placing one replica in each sub-region of the grid. This also aids in reducing the number of replicas within a region and cost of storage capacity. As the replicas can be updated and transmitted, there is no consistency issues and overhead of update propagation. From the experimental analysis, it is witnessed and concluded that the proposed EDPFRS algorithm yields minimum job execution time, network usage and total number of replications. The EDPFRS algorithm requires high storage usage due to the increase in the number of file replicas. Optimization of storage usage, determining and minimizing the cost of file replication is to be considered as the future extension of this work.

References

[1]. R. Mondardini, "Distributed Production Environment for Physics Data Processing," 2006.

[2]. R.-S. Chang and P.-H. Chen, "Complete and fragmented replica selection and retrieval in Data Grids," Future Generation Computer Systems, vol. 23, pp. 536-546, 2007.

[3]. P. Vashisht and A. Sharma, "Decentralized P2P grid resources discovery model in LC-Trie structured overlay," in Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on, 2010, pp. 330-333.

[4]. 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, pp. 233-244, 2011.

[5]. P. Vashisht, R. Kumar, and A. Sharma, "Efficient dynamic replication algorithm using agent for data grid," The Scientific World Journal, vol. 2014, 2014.

[6]. S. S. Sathya and K. S. Babu, "Survey of fault tolerant techniques for grid," Computer Science Review, vol. 4, pp. 101-120, 2010.

[7]. 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, pp. 202-210, 2011.

[8]. S.-M. Park, J.-H. Kim, Y.-B. Ko, and W.-S. Yoon, "Dynamic data grid replication strategy based on internet hierarchy," in International Conference on Grid and Cooperative Computing, 2003, pp. 838-846.

[9]. N. Mansouri and G. H. Dastghaibyfard, "Job scheduling and dynamic data replication in data grid environment," The Journal of Supercomputing, vol. 64, pp. 204-225, 2013.

[10]. H. Shan, L. Oliker, W. Smith, and R. Biswas, "Scheduling in heterogeneous grid environments: The effects of data migration," in International Conference on Advanced Computing and Communication, Gujarat, India, 2004.

[11]. K. Y. Kabalan, W. W. Smari, and J. Y. Hakimian, "Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation," International Journal of Simulation, Systems, Science and Technology, vol. 3, pp. 89-100, 2002.

[12]. Bhuvaneswari R and R. T.N, "IAFCM: Integrated and Adaptive File Consistency Maintenance in Peer-to-Peer Network," IOSR Journal of Engineering (IOSRJEN), vol. 08, pp. 64-75, October 2018.

[13]. N. Saadat and A. M. Rahmani, "PDDRA: A new pre-fetching based dynamic data replication algorithm in data grids," Future Generation Computer Systems, vol. 28, pp. 666-681, 2012.

[14]. N. Mansouri and G. H. Dastghaibyfard, "A dynamic replica management strategy in data grid," Journal of network and computer applications, vol. 35, pp. 1297-1303, 2012.

[15]. U. Tos, R. Mokadem, A. Hameurlain, T. Ayav, and S. Bora, "Dynamic replication strategies in data grid systems: a survey," The Journal of Supercomputing, vol. 71, pp. 4116-4140, 2015.

[16]. Z. Cui and Z. Zhang, "Based on the correlation of the file dynamic replication strategy in multi-tier data grid," International Journal of Database Theory and Application, vol. 8, pp. 75-86, 2015.

[17]. N. Mansouri, "QDR: a QoS-aware data replication algorithm for Data Grids considering security factors," Cluster Computing, vol. 19, pp. 1071-1087, 2016.

[18]. T. Hamrouni, S. Slimani, and F. B. Charrada, "A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids," Engineering Applications of Artificial Intelligence, vol. 48, pp. 140-158, 2016.

[19]. N. Mansouri, "A threshold-based dynamic data replication and parallel job scheduling strategy to enhance Data Grid," Cluster computing, vol. 17, pp. 957-977, 2014.

[20]. J. H. Abawajy and M. M. Deris, "Data replication approach with consistency guarantee for data grid," IEEE Transactions on Computers, vol. 63, pp. 2975-2987, 2014.

[21]. A. M. Rahmani, L. Azari, and H. A. Daniel, "A File Group Data Replication Algorithm for Data Grids," Journal of Grid Computing, vol. 15, pp. 379-393, 2017.

[22]. S. Warhade, P. Dahiwale, and M. Raghuwanshi, "A dynamic data replication in grid system," Procedia Computer Science, vol. 78, pp. 537-543, 2016.

[23]. H. Stockinger, "Database replication in world-wide distributed data grids," Vienna U., 2001.