Journal of ICT, 11, pp: 193–206

DYNAMIC REPLICATION STRATEGY BASED ON EXPONENTIAL MODEL AND DEPENDENCY RELATIONSHIPS IN DATA GRID

Yuhanis Yusof[1], Mohamed Madi[2] and Suhaidi Hassan[3]

School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia

yuhanis@uum.edu.my[1]
S91499@studmail.uum.edu.my[2]
suhaidi@uum.edu.my[3]

ABSTRACT

Data grid is an infrastructure that manages huge amounts of data files, and provides intensive computational resources across geographically distributed collaboration. In order to speed up the data access and reduce bandwidth consumption, data grid replicates essential data in multiple locations. This paper studies the data replication strategy in data grid, taking into account the exponential growth/decay of data files and the dependency relationships between them. Simulation results (via OptorSim) show that the proposed strategy outperformed existing work in the measured metrics – mean job execution time, effective network usage and average storage usage.

Keywords: Grid computing; data grid; data replication; exponential model; dependency level.

INTRODUCTION

Grids become increasingly important to solve computationally data-intensive and time-consuming problems in science, industry, and engineering (Frederic, 2010). Grid computing is an important infrastructure for the purpose of effective utilization of the distributed computational resources (Abdullah, 2004). A data grid (Venugopal, 2006) is a geographically-distributed collaboration in which all members require access to the datasets produced within the collaboration. The databases or datasets of the computational applications in the data grid always have tremendous amounts of data, so the cost of maintaining a local

copy on each site that needs the data is extremely expensive and unrealistic. In other words, such a system requires replica management services that create and manage multiple copies of files. Creating replicas can reroute the clients' requests to certain replica sites and offer a higher access speed than a single server (Tang, 2005).

Data replication is the process of producing multiple copies a data file, and distributing the replicas into grid sites according to certain techniques termed as replication strategies. Replication strategies can be categorized according to their scope of function; replica creation and replica management. Replication creation includes strategies to determine when and where to create a replica of a data file, taking into account of the factors such as request numbers of the data, network conditions, storage availability of nodes. On the other hand, replicas that have been distributed to various locations need to be monitored. Strategies to re-locate replicas, ensuring read write are examples of functions undertaken in replica management.

The replication strategies can also be categorized as static or dynamic. Dynamic strategies (Ben, 2010; Lamehamedi, 2003, Shorfuzzaman; 2008; Zhong, 2010) adapt to changes of data request, bandwidth and storage availability, create replica on new sites or delete the replica dynamically. On the other hand, static strategies will not allow any creation once the data replica number and placement node are chosen (Cibej, 2005; Loukopoulos, 2000). Due to the dynamic characteristics of the data grid, data request frequency from each site varies over time, and the network condition is also not stable. All these factors impact the data access efficiency of the static strategies for they only consider the situations of what the future might be, but cannot adapt to future changes which would happen in the data grid at any time. Dynamic strategies overcome these problems by performing various operations since the changes run at regular intervals or in response to events.

In a data grid, when a data file is required by a job and is not available in the local storage, it may either be replicated or read remotely. If a file is replicated, the next time it is requested, a job can read it quickly and the time to complete the job will be reduced. But, if replicating a data file requires deletion execution of certain jobs (in the future) if may take longer. Therefore, an important decision of determining which data files to be replicated must be made. Identification of the relevant data files can be based on the demand of a file from all the sites in a data grid. So, we need to identify or predict the number of times a data file may be requested in a future-time window (known as replica value).

There are two types of replica requests; a request from the user, i.e. a user directly accesses a file, and a request from a data file, i.e. a file accesses other files. Most of the existing works (Chang, 2008; Tang, 2006; Tang, 2005) focus on the first type of request and ignore the one made by files. Such approaches determine the importance of a file by only tracking the users' requests. This may be applicable if data files in the grid system are running independently, i.e. the files can be executed without invoking other files. But, if the files are running dependently, there is a need to consider the relationships between files in predicting the number of times a file may be requested. In this paper, a dynamic replication strategy that includes information on growth or the decay rate of a file request and the level of file relationships in a data grid is proposed.

The rest of this paper is structured as follows. Section 2 provides a brief description on the existing work in dynamic replication strategies, focusing on how to identify data files that need to be replicated. We include details of our proposed replication strategy in section 3 while the performance evaluation is presented in section 4. Finally, we conclude the work in section 5.

RELATED WORKS

In this section, we introduce some of the studies involving dynamic replication strategies. Two dynamic replication mechanisms (Tang, 2005) are proposed in the multi-tier architecture for data grids, including Simple Bottom-Up (SBU) and Aggregate Bottom-Up (ABU). The SBU algorithm replicates any data file that exceeds a pre-defined threshold. The main shortcoming of SBU is the lack of consideration of the relationship with historical access records. For the sake of addressing the problem, ABU is designed to aggregate the historical records to the upper tier until it reaches the root. Let us consider the data shown in Figure 1. It is an example of the access history for two files, X and Y. In addition, the predefined threshold is 10. According to SBU algorithm, if the parent P1 has enough space, file X will be replicated, since the value of its *numOfAccess* is greater than threshold. In return, file Y will be overlooked, although from the viewpoint of the overall system (looking at the system as a whole) it was accessed 16 times (6 + 10). This means that file Y is more popular compared to file X and therefore should be replicated instead of file X. But SBU algorithm processes the access history individually, and does not consider the relation among the accessed files. On the contrary, Aggregate Bottom Up (ABU) takes into consideration the relation among the files, since it aggregates the files included under the same node, and the file with the highest rate will be replicated. Revert to the last example and apply ABU, the records after aggregation are < P1 , X , 12> and < P1 , Y , 16 >.

nodID	fileID	numOfAccess
C1	X	12
C2	Y	6
C3	Y	10

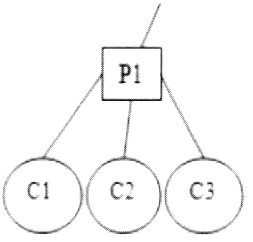

Figure 1. An example of the history and the node relation.

The dynamic replication algorithm proposed in Tang (2006) determines the popularity of a file by analysing the data access history. The researcher believes that the data popular in the past will remain popular in the near future. The history table is in the format of < FID, NOA >, which indicates that the file FID (file ID) has been accessed 6OA (number of access) times. Having analysed the data access history, the average number of access, NOA, is computed. Files with NOA's value that is greater than the computer average NOA will be replicated. Hence, the order of which files to be replicated depends on the NOA. The larger the NOA, the more popular the file is and will be given a higher priority during the replication process.

Nevertheless, these replication strategies did not consider the time period when the files were accessed. If a file was accessed for a number of times in the past, while none was made recently, the file would still be considered popular and hence will be replicated. The algorithm proposed in Chang (2008) called Last Access Largest Weight (LALW) tries to solve this problem. The key point of LALW is to give different weights to files having different ages. The LALW algorithm is similar to other algorithms Tang (2006) in using information on access history to determine the popularity of a file. But innovation is included by adding a tag to each access history record of a file. The weight of the record decays to half of its previous weight after a constant time interval. Older access history records have smaller weights; it means that a more recent historical record is more important. An Access Frequency is calculated to represent the importance of access histories in different time intervals and this is achieved using the formula below.

$$Af(f) = a_0 + \sum_{t=1}^{N_T} \left(a_f^t \times 2^{-(N_T - t)}\right), \forall f \in F$$

Where: N_T is the number of time intervals, F is the set of files that have been requested, and a_f^t indicates the number of accesses for file f at time interval t.

However, this approach (i.e. the LALW system) assumes that the decay rate is constant and equals ½ which means all the files decay at the same rate regardless of the access rate of each one. As a result, the declension rate of weight will be slower. Subsequently the storage element will take time to delete the unwanted files (i.e. the less important files). To address this problem we propose that the value of the file decay varies based on the access rate of the file. This means the decay/growth rate of each file is not the same.

To see the problem of LALW, assume that we have two files; FileA and FileB with corresponding values of Time (T#), number of access (NOA) and Access Frequency (AF), as shown in Table 1:

Table 1

Examples of Access Frequency using LALW

T #	NOA(FileA)	NOA(FileB)	AF(A)	AF(B)
1	40	24	100.0	17.0
2	30	15	110.0	28.5
3	20	20	95.0	38.3
4	10	15	77.5	34.1
5	5	15	58.8	37.1
6	1	10	39.4	33.5

From Table 1, it is learned that number of access of both files (i.e. FileA and FileB) is decreasing but not at the same rate. Based on the six time intervals, the request for FileA decreases more than FileB. On the other hand, at T6, AF(A) is larger than AF(B). According to LALW, FileA is more valuable than FileB, hence it needs to be replicated. However, in the recent time interval (for example T6, T5 and T4), there is less access on FileA as compared to FileB. So, the decision made to replicate FileA can be questioned. Our proposed strategy takes this point into account and suggests that the value of the file decay varies based on the access rate of the file. That means the decay/growth rate of each file is not the same.

THE PROPOSED MODEL (EXPM)

Our replication strategy is designed to include the user's behaviour of requesting a file, and notes the change to this request, whether it is a growth or decay change and also notes the dependency level of the data file (file's behaviour).

Users' Behaviour (File Lifetime)

Many real world phenomena can be modelled by functions that describe how things grow or decay as time passes (Kapitza, 2003; Kremer, 1993). Exponential growth/decay is a positive or negative growth in which the rate of growth is proportional to the current size (Richards, 1959 Bartlett, 1996). This paper proposes to apply the exponential growth/decay rate in determining the importance of a data file. This is due to the fact that each file has its own number of access and this value increases as the access rate increase and vice versa. We describe an exponential model for an access number of files in access history. If we use to represent the number of access for file ***f*** at time *t*, and *N* to represent the number of access at time *t*+1, our exponential growth/ decay model would be:

$$N_f^{t+1} = N_f^t \times (1 + r) \quad (1)$$

Where is the growth or decay rate in the number of access of a file at one time interval. Therefore, we can calculate the value of r using the following formula:

$$r = \left(N_f^{t+1}/N_f^t\right) - 1 \quad (1.1)$$

Assume ***t*** is the number of intervals passed, and indicates the number of access for the file ***f*** at time interval *t*, then we get the sequence of access numbers:

$$N_f^0\ N_f^1\ N_f^2\ N_f^3\ \ldots\ N_f^{t-1}\ N_f^t$$

Therefore, there are time intervals, and each time interval has a growth or decay rate for the number of file access. So, based on Equation 1, the following can be obtained:

$$r_0 = \left(N_f^1/N_f^0\right) - 1,$$
$$r_1 = \left(N_f^2/N_f^1\right) - 1,$$
$$r_{t-1} = \left(N_f^t/N_f^{t-1}\right) - 1\ .$$

Therefore the average rate for all the intervals is as follows:

$$r = \sum_0^{t-1} r_i/t - 1 \quad (1.2)$$

Having known the average accessed rate (growth or decay) for a file during the past time intervals, we can estimate the number of access for data file *f* in the upcoming time interval, *t*, using the following:

$$\text{File Lifetime}\,(t, f) = N_f^t \times (1 + r) \quad (2)$$

In order to avoid extreme cases where the growth or decay rate is equal to infinity, we are assuming that all files have been accessed at least once. Using the same data of *NOA* for *FileA* and *FileB* as provided in Table 1, we obtain the following:

$$\text{File Lifetime}(7, \text{FileA}) = N_{\text{FileA}}^7 = 1 * (1 + (-0.556)) = 0.444$$
$$\text{File Lifetime}(7, \text{FileB}) = N_{\text{FileB}}^7 = 10 * (1 + (-0.291)) = 7.09$$

Based on the values of *File Lifetime (FileA)* and *File Lifetime(FileB)*, it is suggested that at *t=7*, *FileB* will be receiving more request compared to *FileA*.

Files' Behaviour (File Weight)

In a distributed system, there are files that require other files in order to be executed. A file depends on an other file if it needs the later during compilation and/ or execution. The dependency level differs from one file to another. In other words, the importance of a file to the environment is not the same. Our concern is to find the importance of a file to all the files in the system. This is termed as File Weight (FW). The File Weight for file *f* in the upcoming time interval, *t*, can be computed by the following equation:

$$\text{File Weight}(t, f) = \sum_{i=1}^{n} \text{NOA}_i \times \text{DL}_i \quad (3)$$

Where is the Total number of data files in a grid system, NOA is the Number of access at time t-1 for a file that depends on file f, and DL is the dependency level; if there is no dependency then DL is zero.

In order to understand how to calculate File Weight for *FileA* and *FileB* (the example used in Users, Behaviour), consider the data depicted in Figure 2.

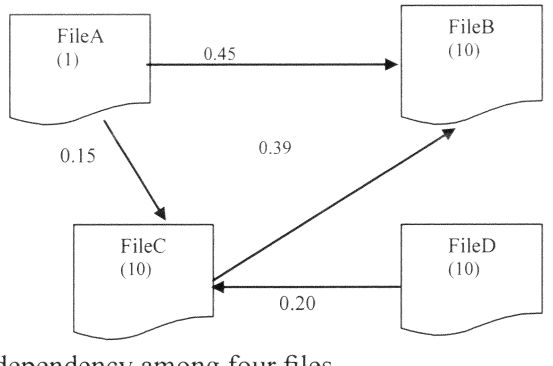

Figure 2. File dependency among four files.

Suppose Figure 2 illustrates the dependency among four files, *FileA*, *FileB*, *FileC* and *FileD* at time *t-1*. The present dependency relationships in Figure 2 would suggest that *FileB* is more important than *FileA* as there are two files (*FileA* and *FileC*) that depend on *FileB* while none exist for *FileA*. Hence, the File Weight of FileA and FileB are obtained as follows:

$$\text{File Weight}(t, f) = \sum_{i=1}^{n} \text{NOA}_i \times \text{DL}_i$$

File Weight(t,FileA) =0x0=0
File Weight(t,FileB)=(1x0.45)+ (10x0.39)=4.35

Our proposed strategy identifies the data files that need to be replicated by combining information from user's behaviour and files behaviour. With this, the File Value is computed as follows:

$$File\ Value(t, f) = FileLifetime(t, f) + FileWeight(t, f) \quad (4)$$

Using Equation 4 to calculate the File Value of *FileA* and *FileB*, we noted that *FileA* is less important than *FileB*. This is due to the values of 0.444 for *FileA* and 11.44 for *FileB*.

EXPERIMENTAL ENVIRONMENT

Dynamic replication algorithms must be tested before deploying them in real data grid environments. A grid simulator called OptorSim (Bell, 2003) which was developed by the European Data Grid project is used to implement and evaluate the proposed algorithm. The study of EXPM was carried out using the model of EU DataGrid sites and their associated network geometry and this is shown in Figure 3.

Jobs were based on the CDF use-case as described in (Huffman, 2002). There were six job types with no overlap between the set of files each job accessed. The simulated EU DataGrid topology used includes 20 sites in the USA and Europe. Within this model, each site, excluding CERN and FNAL, was assigned a Computing and Storage Element. CERN and FNAL were only allocated Storage Elements only, as they produce the original files and store them. The order of files accessed in a job is sequential and is set in the job configuration file as an input to the simulation. The number of files in our simulation was 150, and a file size was 1GB.It is important to understand how the replication algorithms perform with increasing numbers of jobs on the grid. Thus the number of jobs that were considered in our evaluation varied between 200 and 4000 jobs. The basic settings and parameters used in this experiment are shown in Table 2.

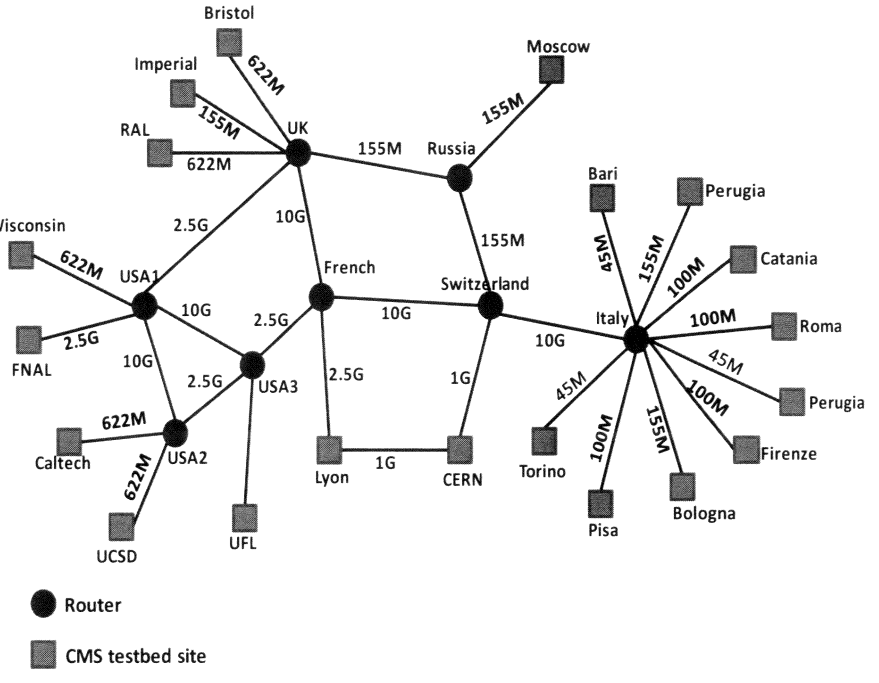

Figure 3. The EU data grid testbed sites and their associated network geometry.

Table 2

Parameters Settings in OptorSim

Parameter	*Value*
Number of jobs	*200, 500, 1000, 2000 and 4000*
Scheduler	*QAC scheduler*
Site policy	*All job types*
Access history length	*1000000 ms*
Storage metric	*0.67*
Maximum queue size	*200*
Job delay	*2500 ms*

Simulation Results

The performance metrics we chose to evaluate the proposed system were: Mean Job Execution Time (MJET), Efficient Network Usage (ENU), and Average Storage Usage (ASU). MJET is the average time taken to execute a job, from the moment it is scheduled to the Computing Element to the moment it has finished processing all the required files. ENU (Cameron, 2004, Bell, 2003) is used to estimate the efficiency of the network resource usage. A lower value indicates that the utilization of the network bandwidth is more efficient. ASU is a percentage of the capacity reserved by the files according to the total capacity for the underlying storage. The proposed strategy (EXPM) is compared against the Simple Optimizer, the Least Frequently Used (LFU) algorithm that is already implemented in OptorSim, and LALW (Last Access Largest Weight). The Simple Optimizer is a base case which does not involve any replication and files are accessed remotely. On the other hand, the LFU algorithm always replicates data files. The LALW algorithm is presented in (Chang, 2008).

Based on Figure 4, there is a linear increase in MJET as the number of jobs on the grid increases. This is because, as more jobs are submitted, the queues at the sites increase. If the job submission rate is higher than the grid's job processing rate, this build-up of queues is inevitable, and it is likely that this would also occur in a real grid. A better system is the system that produces less MJET. In the undertaken experiment, EXPM performs the best among the existing systems when evaluated based on MJET. The EXPM has the least mean job execution time and is 5.12 per cent faster than LALW, and about 7 per cent over LFU.

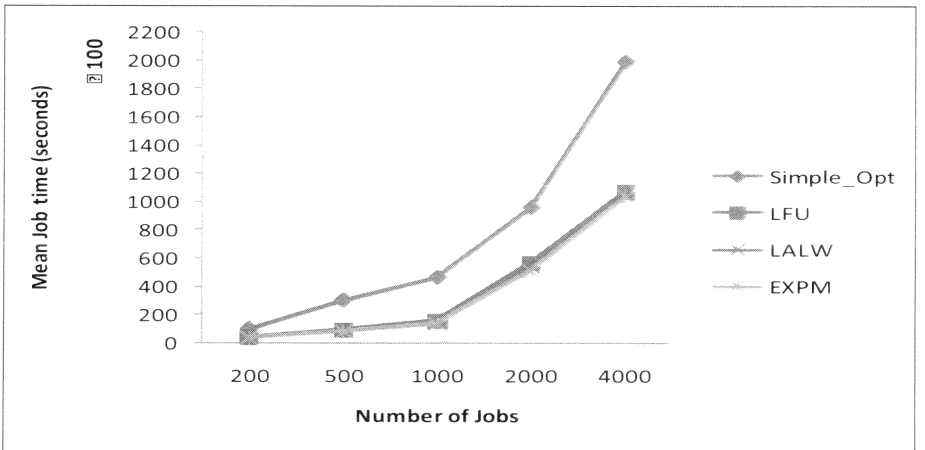

Figure 4. Mean job execution time.

On the other hand, LFU uses the highest amount of storage (ASU) as they always replicate the files to the local storage element. This is followed by the LALW system. Figure 5 clearly shows that by using EXPM the average storage usage can be reduced. EXPM outperforms LFU and LALW by 18.02 per cent and 14.51 per cent respectively.

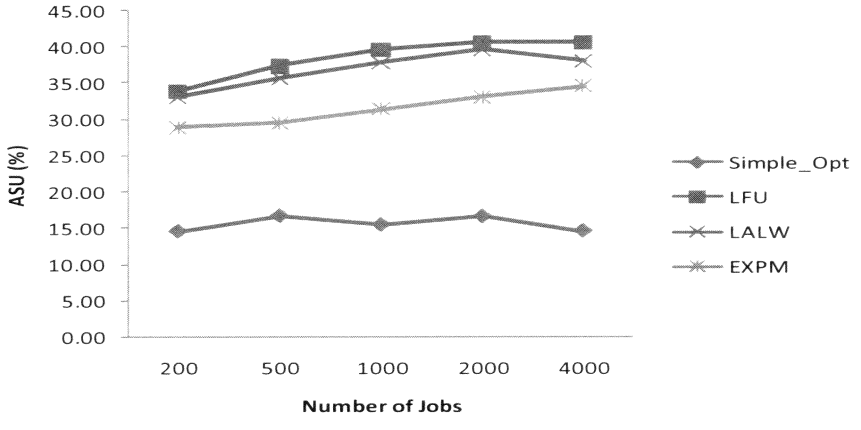

Figure 5. Storage resources usage.

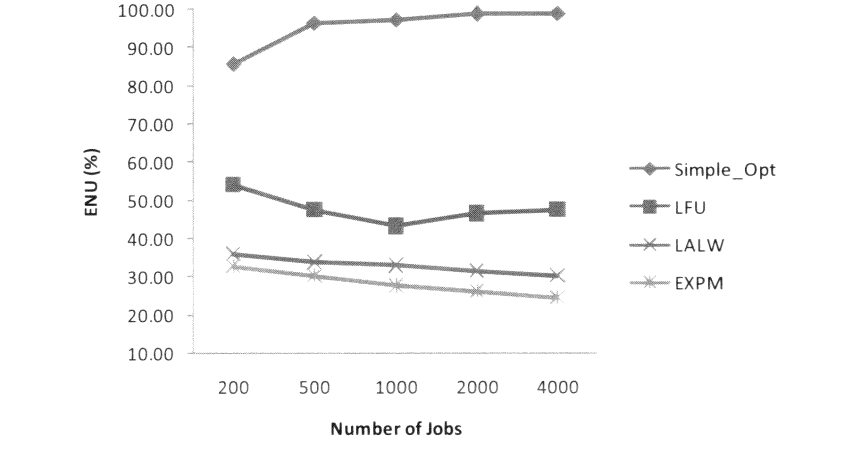

Figure 6. Efficient network usage.

The results of ENU metric (Figure 6) show a slight linear decrease as the number of jobs on the grid increases. This is because at the start of the simulation, the queues are small, but they build up quickly while the files are

copied around the grid. Once the replication process has established, the jobs can run faster and hence the queues decrease. The efficient network usage decreases with increasing numbers of jobs because the amount of replication decreases over time. Simple Optimiser and LFU have the highest efficient network usage, showing that they are poor at making replication decisions. The EXPM uses the lowest amount of network resources for all numbers of jobs because it makes better decisions on which file should be replicated.

Data in Table 3 depicts the average values for the undertaken experiments (of different numbers of jobs). It can be seen that the EXPM strategies produce the best result in mean job-execution time and efficient network usage. Such a result is obtained as EXPM is able to identify better which files need to be replicated. Hence, producing the least job-execution time and using the least amount of network. As for the Average Storage Usage, EXPM comes second after Simple Optimizer and this is due to the fact that Simple Optimizer only reads files remotely, hence storing less files in its storage element.

Table 3

Simulation Results of EXPM and Existing Algorithms

Metrics	Simple Optimizer	LFU	LALW	EXPM
MJET	76004	38796	37852	35915
ENU	95.47	47.75	32.83	28.19
ASU	15.60	38.46	36.88	31.53

CONCLUSION

Exponential growth and decay are mathematical changes. The rate of change continues to either increase or decrease as time passes. In this paper we adopted the exponential model and noted the growth or decay rate of file requests from users. In addition, we also used information of file dependency to help in deciding which data files to replicate. Such an approach identifies the importance of a data file from both points of views; users and the files system in the grid. Simulation results (via OptorSim) show that the proposed strategy, EXPM, outperformed LALW in the measured metrics – mean job-execution time, effective network usage and average storage usage. For future work, we plan to extend our model to include decisions on replica deletions so that extra storage space can be created to store on demand data files.

REFERENCES

Abdullah, A., & Sulaiman, N. (2004). A simulation study of data discovery mechanism for scientific data grid environment. *Journal of Information Communication Technology*, *3*(1), 19-32.

Bartlett, A. A. (1996). The exponential function. XI: The new flat earth society the physics teacher, *34*, 342-343.

Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., & Zini, F. (2003). *Evaluation of an economy-based file replication strategy for a data grid.* Paper presented at the International Workshop on Agent based Cluster and Grid Computing, 120-126.

Ben Charrada, F., Ounelli, H., & Chettaoui, H. (2010). *An efficient replication strategy for dynamic data grids*. Paper presented at the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 50-54.

Cameron, D. G., Millar, A. P., Nicholson, C., Carvajal-Schiaffino, R., Stockinger, K., & Zini, F. (2004). Analysis of scheduling and replica optimisation strategies for data grids using OptorSim. *Journal of Grid Computing*, *2*(1), 57-69.

Chang, H. P. (2008). A dynamic weighted data replication strategy in data grids. Paper presented at the AICCSA 2008. *Proceedings of IEEE/ ACS International Conference on computer systems and applications*, 414-421.

Cibej, U., Slivnik, B., & Robic, B. (2005). The complexity of static data replication in data grids. *Parallel Computing*, *31*(8-9), 900-912.

Frederic Magoulès. (2010). *Fundamentals of grid computing: Theory, algorithms and technologies*. USA: Chapman & Hall/CRC Numerical Analysis & Scientific Computing.

Huffman, B. T., McNulty, R., Shears, T., Denis, R. S., & Waters, D. (2002). *The CDF/D0 UK GridPP Projec*t. CDF Internal Note, 5858.

Kapitza, S. P. (2003). The statistical theory of global population growth. *Formal descriptions of developing systems*.

Kremer, M. (1993). Population growth and technological change: One million BC to 1990. *Quarterly Journal of Economics-Cambridge Massachusetts-*, 108, 681-681.

Lamehamedi, H., Shentu, Z., Szymanski, B., & Deelman, E. (2003). *Simulation of dynamic data replication strategies in data grids*. Paper presented at the 12th Heterogeneous Computing Workshop (HCW2003).

Loukopoulos, T., & Ahmad, I. (2000). *Static and adaptive data replication algorithms for fast information access in large distributed systems*.

Richards, F. J. (1959). A flexible growth function for empirical use. *Journal of Experimental Botany*, *10*(2), 290-301.

Shorfuzzaman, M., Graham, P., & Eskicioglu, R. (2008). *Popularity-driven dynamic replica placement in hierarchical data grids*. Paper presented at the Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008, 524-531.

Tang, M., Lee, B. S., Tang, X., & Yeo, C. K. (2006). The impact of data replication on job scheduling performance in the data grid. *Future Generation Computer Systems*, *22*(3), 254-268.

Tang, M., Lee, B. S., Yeo, C. K., & Tang, X. (2005). Dynamic replication algorithms for the multi-tier data grid. *Future Generation Computer Systems*, *21*(5), 775-790.

Venugopal, S., Buyya, R., & Ramamohanarao, K. (2006). A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, *38*(1).

Zhong, H., Zhang, Z., & Zhang, X. (2010). *A dynamic replica management strategy based on data grid*. Paper presented at the 2010 Ninth International Conference on Grid and Cloud Computing, 18-23.