A DYNAMIC REPLICA CREATION: WHICH FILE TO REPLICATE?

Mohammed Madi[1], Yuhanis Yusof[2], Suhaidi Hassan[3]

[1] Universiti Utara Malaysia, Malaysia , s91499@studmail.uum.edu.my
[2, 3] Universiti Utara Malaysia, Malaysia,{yuhanis,suhaidi}@uum.edu.my

ABSTRACT. Data Grid is an infrastructure that manages huge amount of data files and provides intensive computational resources across geographically distributed collaboration. To increase resource availability and to ease resource sharing in such environment, there is a need for replication services. Data replication is one of the methods used to improve the performance of data access in distributed systems. In this paper, we propose a dynamic replication strategy that is based on exponential growth or decay rate and dependency level of data files (EXPM). Simulation results (via Optorsim) show that EXPM outperformed LALW in the measured metrics – mean job execution time, effective network usage and average storage usage.

Keywords: Data Grid, resource sharing, data replication, EXPM

INTRODUCTION

A Data Grid (Venugopal, 2006) is a geographically-distributed collaboration in which all members require access to the datasets produced within the collaboration. In Data Grids (Foster, 2001)(Foster, 2002), distributed scientific and engineering applications often require access to a large amount of data or they continuously generate several terabytes, even petabytes, of raw data in data grid. Therefore one of the tasks in Data Grid is to manage the huge amount of data and facilitate data and resource sharing. In order to achieve this task, data must be copied and stored in several physical locations to vouch the efficient access, without a large consumption of the bandwidth and access latency. In other words, such a system requires replica management services that create and manage multiple copies of files. Creating replicas can reroute the client requests to certain replica sites and offer a higher access speed than a single server (Tang, 2005)

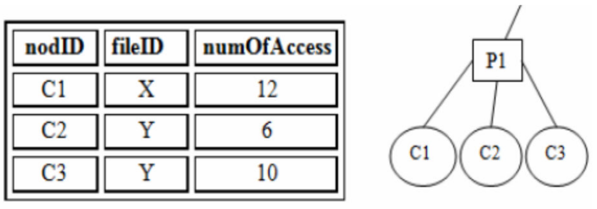
In a Data Grid, when a file is required by a job and is not available on local storage, it may either be replicated or read remotely. If a file is replicated, the next time it is requested, the job can read it quickly and the time to complete the job will be reduced. But, if replicating a file requires deletion of other files, execution of certain jobs (in the future) may take longer. Therefore, an important decision of determining files to be replicated must be made. Replica value is defined as the number of times a replica will be requested in a future time window. There are two types of replica request; a direct request from user, i.e. a user directly access a file, and an indirect request from a file, i.e. a file accesses other files by calling one or more of its methods. Most of the existing works (Chang, 2006) (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 users' request. This may be applicable if files in Data Grid system are running independently, i.e. files can be executed without invoking other files. But, if files are running dependently, there is a need to assume both direct and indirect requests. In this paper, a dynamic replication strategy, known as Exponential

Mechanism (EXPM), is designed by tracking both the users and file behavior. Outcome of this strategy is the identification of files to be replicated.

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

RELATED WORKS

In this section, we introduce some of the studies undertaken 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 to 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 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. On the other hand, file Y will be overlooked, although from the viewpoint of the overall system (looking the system as a whole) it was accessed for 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. In 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 same 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 popularity of a file by analyzing data access history. The researcher believes that the popular data 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 NOA (number of access) times. Having analyzed 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 time period of 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, 2006) 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 age. The LALW algorithm is similar to other algorithms (Tang, 2006) by means of using information on access history to determine popularity of a file. But, the 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 stated as 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. LALW system) assumes that the decay rate is constant and equals ½ that means all of files decay in the same rate regardless 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 file growth/decay varies based on the access rate of the file. That means the growth/decay rate of each file is not the same.

THE PROPOSED MODEL

Our replication system is designed by integrating information on file popularity from two perspectives; users and the file system. The first viewpoint is based on users behavior of requesting a file while the latter utilize information on dependencies of files in the grid system.

Users' Behavior of Requesting a File (File Lifetime)

Many real world phenomena can be modeled 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 work proposes to apply the exponential growth/decay rate in determining importance of a file (Madi, 2009). We describe an exponential growth/decay model for file's number of access in access history. The process of accessing files in data grid environment follows an exponential model. If we use $\mathrm{N_f^t}$ to represent the number of access for file f at time t, and $\mathrm{N_f^{t+1}}$ to represent the number of access at time $\mathrm{t+1}$, our exponential growth/decay model would be given by:

$$\mathrm{N_f^{t+1}} = \mathrm{N_f^t} \times (1 + \mathrm{r}) \qquad (1.1)$$

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

$$\mathrm{r} = \left(\mathrm{N_f^{t+1}}/\mathrm{N_f^t}\right) - 1 \qquad (1.1.1)$$

Assume t is the number of intervals passed, and $\mathrm{N_f^t}$ indicates the number of access for the file f at time interval t, then we get the sequence of access numbers:

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

Therefore, there are $\mathrm{t-1}$ time intervals, and each time interval has a growth or decay rate in number of access of a file. So, according to the exponential growth/decay model we can write:

$$\mathrm{r_0} = \left(\mathrm{N_f^1}/\mathrm{N_f^0}\right) - 1,\ \mathrm{r_1} = \left(\mathrm{N_f^2}/\mathrm{N_f^1}\right) - 1,\ \mathrm{r_2} = \left(\mathrm{N_f^3}/\mathrm{N_f^2}\right) - 1,$$
$$\mathrm{r_{t-1}} = \left(\mathrm{N_f^t}/\mathrm{N_f^{t-1}}\right) - 1 \qquad (1.1.2)$$

Therefore the average rate for all intervals is $\mathrm{r} = \sum_0^{t-1} \mathrm{r_i}/\mathrm{t} - 1$ (1.1.3)

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

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

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 for at least once.

Files Behavior of Requesting a File (File Weight)

In a distributed system, there are files that require other files in order to be executed - dependency level of a file. A file depends on 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 files in the system. This is termed as File Weight (FW). The File Weight is computed by the following equation:

$$\text{File Weight} = \sum_{i=1}^{n} \text{NOA}_i \times \text{DL}_i \tag{1.2}$$

Where, n: Total number of the files in the grid system, NOA: Number of access of the file that needs the underlying file, DL: The dependency level of the file, if there is no dependency then the DL is zero.

In order to understand how to calculate File Weight, consider the following example: Suppose that we have four files in a grid system: File1, File2, File3 and File4. The DL and NOA for the files are shown in Figure 2.

$$File\ Weight(File_1) = 0$$

$$File\ Weight(File_2) = (20 * 0.45) + (15 * 0.39) = 14.85$$

$$File\ Weight(File_3) = (20 * 0.15) + (30 * 0.20) = 9$$

$$File\ Weight(File_4) = 0$$

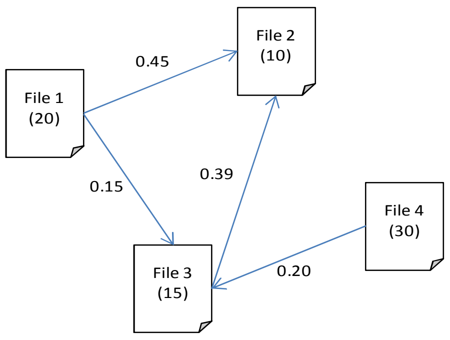


Figure 2: An example files dependency

Based on Figure 2, File 2 has the highest weight among the files, which means File 2 is the most important file for the current grid system. FW And FL are used to compute the $File\ Value$, that indicates the volume of demand on a file in the grid system. The larger $File\ Value$,is for a file, the more popular is the file. Hence, it needs to be replicated. $File\ Value$ is computed by the following equation:

$$File\ Value = FL + FW \tag{1.2.1}$$

EXPERIMENTAL ENVIRONMENT

Dynamic replication algorithms must be tested before deploying them in real Data Grid environments. A Grid simulator that is called OptorSim (Bell, 2003) which was developed by the European Data Grid project is used in order to implement and evaluate the proposed algorithm. The topology of our simulated platform adapts the topology and configuration used in (Chang, 2006) as it is the most similar work to ours. This configuration has four clusters and each one has three sites. One site has the most capacity in order to hold all the master files at the beginning of the simulation. The others have a uniform size, 50GB. All the network bandwidth is set as 100 Mbits/sec. The connection bandwidth is 100 Mbps. There are 500 jobs need to be submitted. We ran the simulation with 500 jobs. A job is submitted to Resource Broker every 25 second. Resource Broker then submits to Computing Element according to an appropriate scheduling algorithm. There are 6 job types, and each job type requires specific files for execution. 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 is 150, and a file size is 1GB.

Simulation Results

The performance metrics we chose to evaluate the proposed system are: Mean Job Execution Time (MJET), Efficient Network Usage (ENU), and Average Storage Usage (ASU). MJET is the average time a job takes to execute, from the moment it is scheduled to Computing Element to the moment when it has finished processing all the required files. ENU (Cameron, 2004, Bell,2003) is used to estimate the efficiency of network resource usage. A lower value indicates that the utilization of network bandwidth is more efficient. ASU is a percentage of capacity consumed by files over the total capacity for the underlying storage. The proposed model (EXPM) is compared against the Simple Optimizer and LALW (Last Access Largest Weight). The Simple Optimizer is a base case which does not involve any replication and files are accessed remotely. The LALW algorithm is as presented in (Chang, 2006).

A summary of the results is shown in Table 1. As shown in Figure 3, the mean job execution time using EXPM is about 22% faster than Simple optimizer, and 5% than LALW. Figure 4 illustrates the ENU metric of the three strategies. The Simple Optimizer consumes the most network bandwidth as CEs need to read all files remotely. However, LALW and EXPM reduce the bandwidth consumption by half. Moreover, EXPM outperforms LALW by 9% in improving ENU. This is because number of replications required by EXPM is less LALW - EXPM depends on two criteria to determine files that require replication as compared to only one by LALW. Figure 5 illustrates the storage value of the strategies - Simple Optimizer uses the least amount of storage while the EXPM outperforms LALW by 7%. This is because in EXPM, the base of exponential decay varies based on the access rate of the file. Contrary to LALW approach which assumes that the base of exponential decay is constant and equals ½ - all files decay in the same rate regardless of its access rate. As a result, the declension rate of weight will be slower.

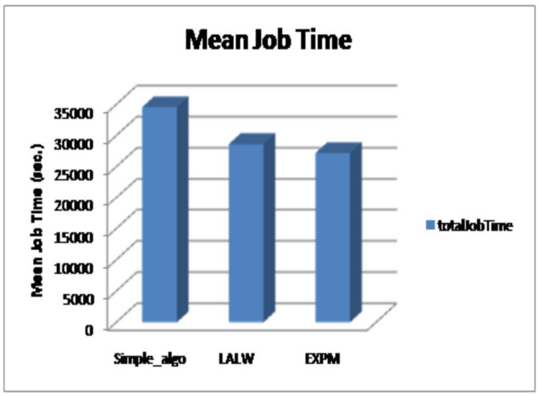


Figure 3: Mean job execution time

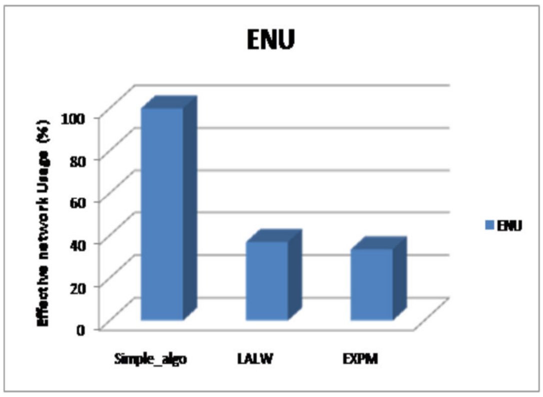


Figure 4: Effective Network Usage

Table 1: Simulation results of LALW and EXPM

	Simple_algo	LALW	EXPM
ASU	10.42	23.32	21.69
ENU	100.00	36.87	33.43
Mean Job Time	34573.52	28626.88	27195.53

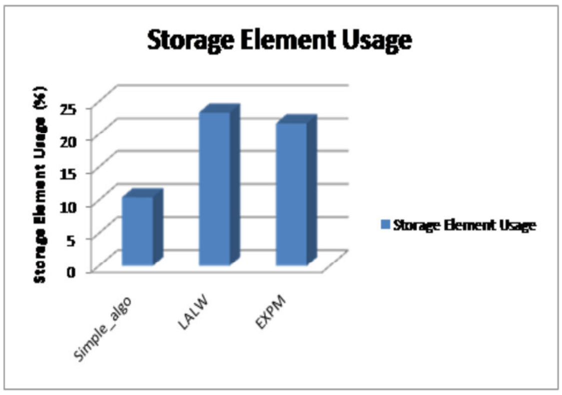


Figure 5: Storage Resources Usage

CONCLUSION

Exponential growth and decay are mathematical changes. The rate of the change continues to either increase or decrease as time passes. In this paper we adopted the exponential growth and decay, and file dependency in determining files that need to be replicated. Such an approach considers both the user and file behaviors. 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 decision on replica deletion by investigating approaches to determine the minimum and maximum threshold to categorize popularity of files.

REFERENCES

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., Capozza, L., Millar, A. P., Stockinger, K., & Zini, F. (2002). Simulation of Dynamic Grid Replication Strategies in OptorSim. Lecture notes in computer science, 46-57.

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.

Bonhoeffer, S., May, R. M., Shaw, G. M., & Nowak, M. A. (1997). Virus dynamics and drug therapy (Vol. 94, pp. 6971-6976): National Acad Sciences.

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. (2006). A Dynamic Data Replication Strategy Using Access-Weights in Data Grids.

Chang, R. S., & Chen, P. H. (2007). Complete and fragmented replica selection and retrieval in Data Grids. Future Generation Computer Systems, 23(4), 536-546.

Foster, A. C. I., Salisbury, C. K. C., & Tuecke, S. (2001). The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, 23.

Foster, I. (2002). The Grid: A New Infrastructure for 21st Century Science. Physics today, 55(2), 42, 47.

Foster, I., & Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure. 1999. San Francisco: Morgan Kaufmann Publishers, 24(677), 8.

Goel, S., & Buyya, R. (2007). Data replication strategies in wide area distributed systems. Enterprise Service Computing: From Concept to Deployment.

Hunt, R., & Institute of Terrestrial, E. (1978). Plant Growth Analysis.

Kapitza, S. P. (2003). The statistical theory of global population growth. Formal Descriptions of Developing Systems.

Kreft, J. U., Booth, G., & Wimpenny, J. W. (1998). BacSim, a simulator for individual-based modelling of bacterial colony growth. Microbiology, 144(12), 3275-3287.

Kremer, M. (1993). Population Growth and Technological Change: One Million BC to 1990. Quarterly journal of economics-cambridge massachusetts-, 108, 681-681.

Lamehamedi, H., Szymanski, B., Shentu, Z., & Deelman, E. (2002). Data Replication Strategies in Grid Environments.

Madi. M., Yusof Y. Hassan S. (2009). A Dynamic Replication Strategy based on Exponential Growth/Decay Rate. Proceedings of the 2nd International Conference on Computing and Informatics, 48-53.

Newling, B. E. (1974). Urban Growth and Spatial Structure: Mathematical Models and Empirical Evidence. Comparative Urban Structure.

Ranganathan, K., & Foster, I. (2001). Design and Evaluation of Dynamic Replication Strategies for a High Performance Data Grid.

Ranganathan, K., & Foster, I. (2001). Identifying Dynamic Replication Strategies for a High-Performance Data Grid. Lecture notes in computer science, 75-86.

Richards, F. J. (1959). A Flexible Growth Function for Empirical Use. Journal of Experimental Botany, 10(2), 290-301.

Stockinger, H., Samar, A., Holtman, K., Allcock, B., Foster, I., & Tierney, B. (2002). File and Object Replication in Data Grids. Cluster Computing, 5(3), 305-314.

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. (2006). Scheduling Distributed Data-intensive Applications on Global Grids: University of Melbourne, Dept. of Computer Science and Software Engineering.

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).