A Dynamic Different Threshold Strategy to Data Replication in Data Grids

Ye-In Chang
National Sun Yat-Sen University,
Taiwan, R.O.C.
Email:changyi@cse.nsysu.edu.tw

Lee-Wen Huang
National Sun Yat-Sen University,
Taiwan, R.O.C.
and
Far East University,
Taiwan, R.O.C.
Email:huanglw@gmail.com

Yen-Wei Huang
National Sun Yat-Sen University,
Taiwan, R.O.C.
Email:huangyw@db.cse.nsysu.edu.tw

***Abstract*—In the context of data grid technology, data replication is mostly used to reduce access latency and bandwidth consumption. Two of well-known replication strategies are $Fast\text{-}Spread$ and $Cascading$, which work well for different kinds of access patterns individually. Later, Chang $et\ al.$ propose the static different threshold (DT) strategy, which can work for any kind of access patterns. However, among a large number of different data files, there may exist some hot data files which have been most requested. To reduce the number of requests for the hot files, we should replicate those hot files as soon as possible. Therefore, in this paper, we propose the dynamic DT strategy. In the dynamic DT strategy, data replication of hot files occurs earlier than others because the thresholds of hot files are decreased earlier than thresholds of the normal files. From our simulation results, we have shown that the performance of the dynamic DT strategy is better than the previous strategies.**

***Index Terms*—Access pattern, Data Grid, Data replication, Dynamic different threshold, Grid**

I. Introduction

The world of Grid computing is continuously growing, and new projects are founded in an increasing rate. These projects range from dedicated grid computing infrastructures to public infrastructures, from academic to commercial, from pilot projects to production systems, and from proof-of-concept to traditional science applications [7], [13].

Grid systems are inter connected collections of heterogeneous and geographically distributed resources harnessed together to satisfy various needs of the users. Resource management is the central component of a grid system. It involves managing resources in the system. Its basic responsibility is to accept requests from users, match user requests to available resources for which the user has accessed and scheduled the matched resources [12].

Initially, the Grid was envisioned as a way to share large computing facilities, sending jobs to be executed at remote computational sites. For this reason, the Grid was referred to as a *Computational Grid*. However, very large jobs are often data intensive, and in such cases, it may be necessary to move the job to where the data sites are in order to achieve better efficiency. Thus, the term *Data Grid* was used to emphasize applications that produce and consume large volumes of data [1], [5].

Because of the distribution and large amount of data, even more than Petabytes (2^{20} Gigabytes), there would be several kinds of access patterns. The access pattern which is generated randomly is the worst-case scenario, while most of realistic access patterns contain varying amounts of temporal and geographical locality. *Temporal Locality* means that the files recently accessed are likely to be accessed again. *Geographical locality* (Client locality) means that the files recently accessed by a client are likely to be accessed by nearby clients.

When a user generates a request for a file, large amounts of bandwidth would be consumed to transfer the large data file from the server to the client. Furthermore, the latency involved could be significant considering the size of the files involved. Ranganathan and Foster [9] have investigated the usefulness of creating replicas to distribute these huge data sets among the various scientists in Data Grids. So, the replication strategies are used to reduce the cost in Data Grids. The main aims of using replication are to reduce access latency and bandwidth consumption. The other advantages of replication are that it helps in load balancing and improves reliability by creating multiple copies of the same data. Static replication can be used to achieve some of the above-mentioned gains, but the drawback with static replication is that it cannot adapt to changes of the user behavior. The replicas have to be manually created and managed, if one were to use static replication. In their scenario, where the data amounts to petabytes (2^{20} Gigabytes), and the user community is in the order of thousands around the world, static replication does not sound feasible [9]. Such a system needs dynamic replication strategies, where replica creation, deletion and management are done dynamically depending on the situations of the requests and the capacity of each node. The data in this scenario is read-only, and so there are no consistency issues involved. Dynamic strategies have the ability to adapt to changes in user behavior. Ranganathan and Foster have proposed several different dynamic replication strategies for Data Grids [9], [10], which can work well under different access patterns individually, including $Cascading$ and $Fast\text{-}Spread$. If the grid users exhibit total randomness in accessing data, then the $Fast\text{-}Spread$ strategy can work best among their proposed strategies. If, however, there were sufficient amount of geographical locality in the access patterns, the $Cascading$ strategy can work better than the others [9], [10].

However, in the real world, there are many unpredictable access patterns. Therefore, Chang *et al.* [2] propose to use only one strategy which can work for any kind of access patterns. They propose a static *Different Threshold (DT)* strategy for data replication in Data Grids, which could be dynamically adapted to several kinds of access patterns and achieve different replication effects. In the static DT strategy, each layer has its own threshold, whereas the threshold in each layer of Ranganathan and Foster's strategies is always the same. When the number of requests for the files from a node reaches the threshold of its layer, the file will be replicated to the node. For the same logic network topology, they can use only one way to achieve different replication effects. Moreover, the static DT strategy can provide even better performance than the $Cascading$ and $Fast\text{-}Spread$ strategies by setting the threshold T_i suitably, where T_i is the threshold of ith layer of the tree.

However, among a large number of different data files, there may exist some hot data files. That is, the difference in the number of requests between each data file may be very large. Those files which have been requested most often are hot data files. Therefore, in this paper, we propose a dynamic DT strategy. In the static DT strategy, we set different threshold T_i at each layer i before replicating, and the thresholds at each layer i are fixed. In the dynamic DT strategy, besides the static thresholds at each layer, we set an offset to the threshold of each data file; that is, each data file has its own threshold, and it may be adjusted while the replication of the data file is occurring. To reduce the number of requests for the hot files, we let data replication of hot files occur earlier than others by decreasing the thresholds of hot files earlier than the normal ones. Therefore, in the simulation study, we show that the performance of the dynamic DT strategy is better than that of the static DT strategy.

The rest of the paper is organized as follows. In Section 2, we give a survey of some well-known Data Grids strategies and architectures. In Section 3, we present the dynamic *Different Threshold (DT)* strategy. In Section 4, we compare the performance of our strategy with the previous strategies. Finally, we give a conclusion in Section 5.

II. Background

Data replication has been studied extensively, and different distributed replica management strategies have been proposed in the literature [3], [11]. The techniques of Data Grids facilitate the distribution of such data to geographically remote places. Dynamic replication can be used as a technique to reduce response time, bandwidth consumption and access latency in accessing these huge amounts of data. Ranganathan and Foster [9] evaluate the performance of six different replication strategies for three different kinds of access patterns.

Furthermore, Ranganathan and Foster have proposed several strategies [10] which can work

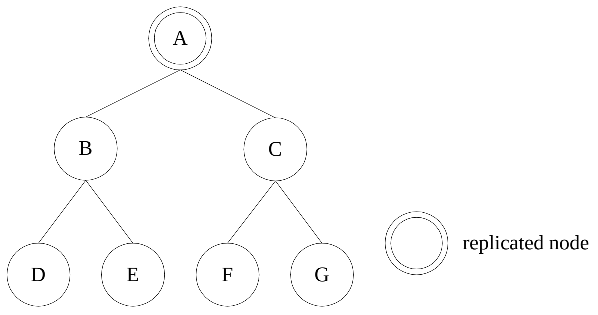

Fig. 1. The Cascading strategy

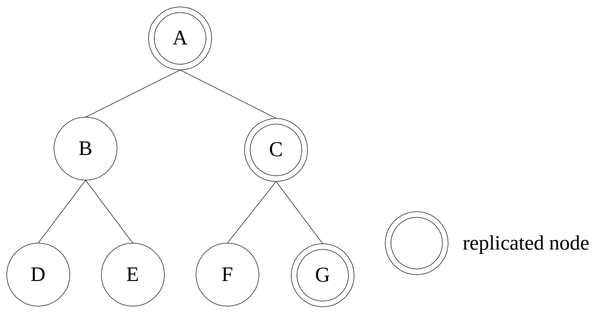

Fig. 2. The Fast Spread strategy

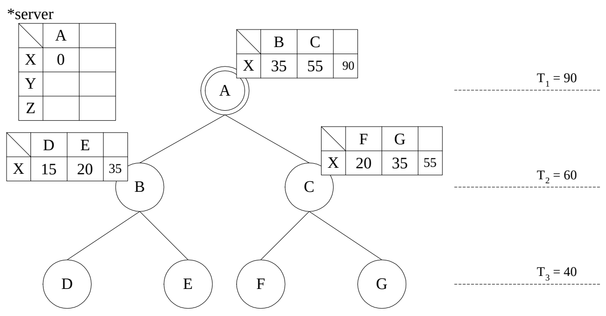

Fig. 3. The similar effect of the Cascading strategy

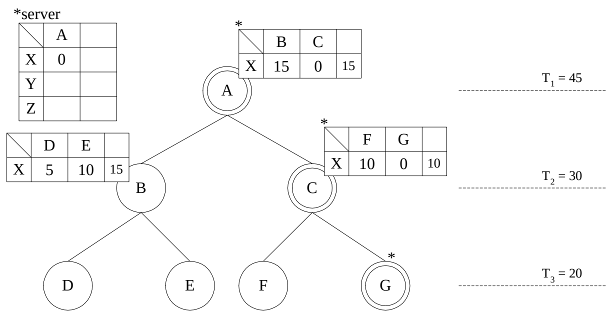

Fig. 4. The similar effect of the Fast-Spread strategy

well under different access patterns individually, including $Cascading$ and $Fast$-$Spread$. In the $Cascading$ strategy, once the threshold for a file is exceeded at the root, a replica is created at the next level, as shown in Figure 1. Using the $Fast$-$Spread$ strategy, a replica of the file is stored at each node along its path to the client, as shown in Figure 2. If the grid users exhibit total randomness in accessing data, the $Fast$-$Spread$ strategy can work best. If, however, there were sufficient amount of geographical locality in the access patterns, the $Cascading$ strategy can work better than the others.

However, in the real world, there are many unpredictable access patterns. Therefore, Chang *et al.* [2] propose to use only one strategy which can work for any kind of access patterns. They propose a static *Different Threshold (DT)* strategy for data replication in Data Grids, which could be dynamically adapted to several kinds of access patterns and achieve different replication effects. In the static DT strategy, each layer has its own threshold, whereas the threshold in each layer of Ranganathan and Foster's strategies is always the same. When the number of requests for the files from a node reach the threshold of its layer, the file will be replicated to the node. For the same logic network topology, they can use only one way to achieve different replication effects as shown in Figure 3 and Figure 4, respectively, where T_i is the threshold of ith layer of the tree.

Moreover, the static DT strategy can even provide better performance than the $Cascading$ and $Fast$-$Spread$ strategies by setting the threshold T_i suitably. Figure 5 and Figure 6 show the comparison of three strategies: *Fast-Spread*, *Cascading*, and DT on the *P–random* and *P–gt* access patterns. In these figures, we assume that the leaf nodes can send the requests for data. Then we account and record the number of requests from each leaf node until the replication is done. The number recorded at each node means the number of requests from the node. The number which is underlined means that it will not be increased any more. So we can see that the number of requests needed in the strategy DT is less than that needed in the other two strategies. It means that the DT strategy can provide the better performance than the $Cascading$ and $Fast$-$Spread$ strategies.

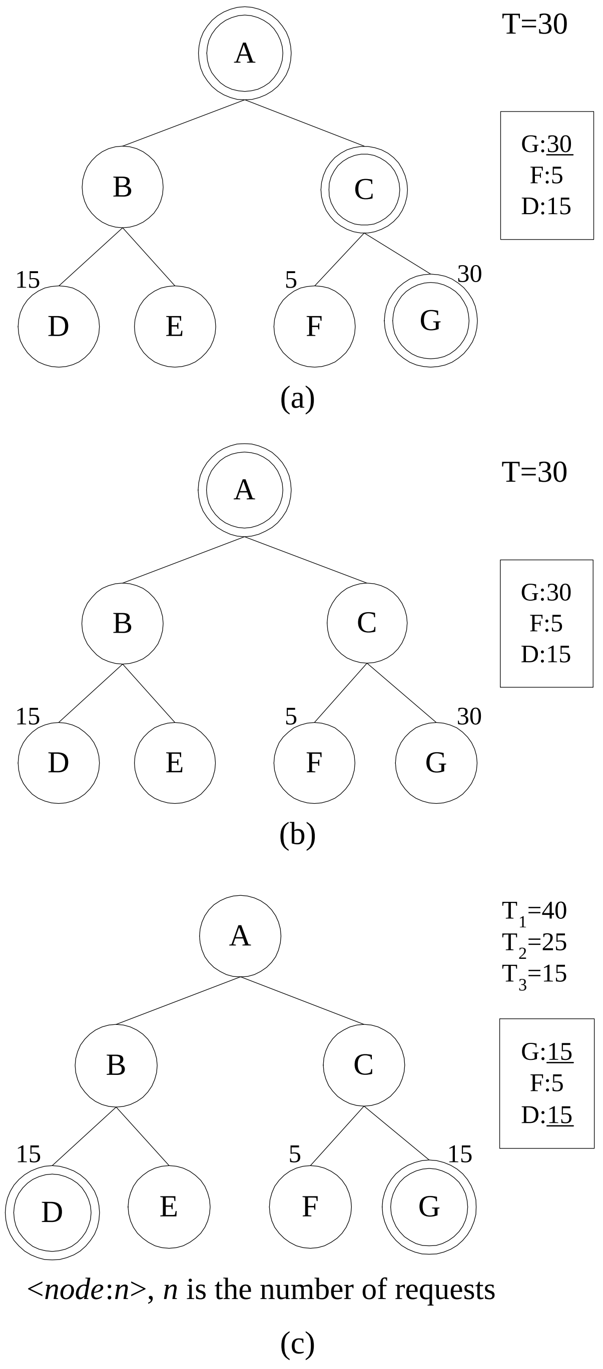

Fig. 5. A comparison on the random access pattern: (a) the Fast-Spread strategy with T = 30; (b) the Cascading strategy with T = 30; (c) the DT strategy with T_1 = 40, T_2 = 25, T_3 = 15.

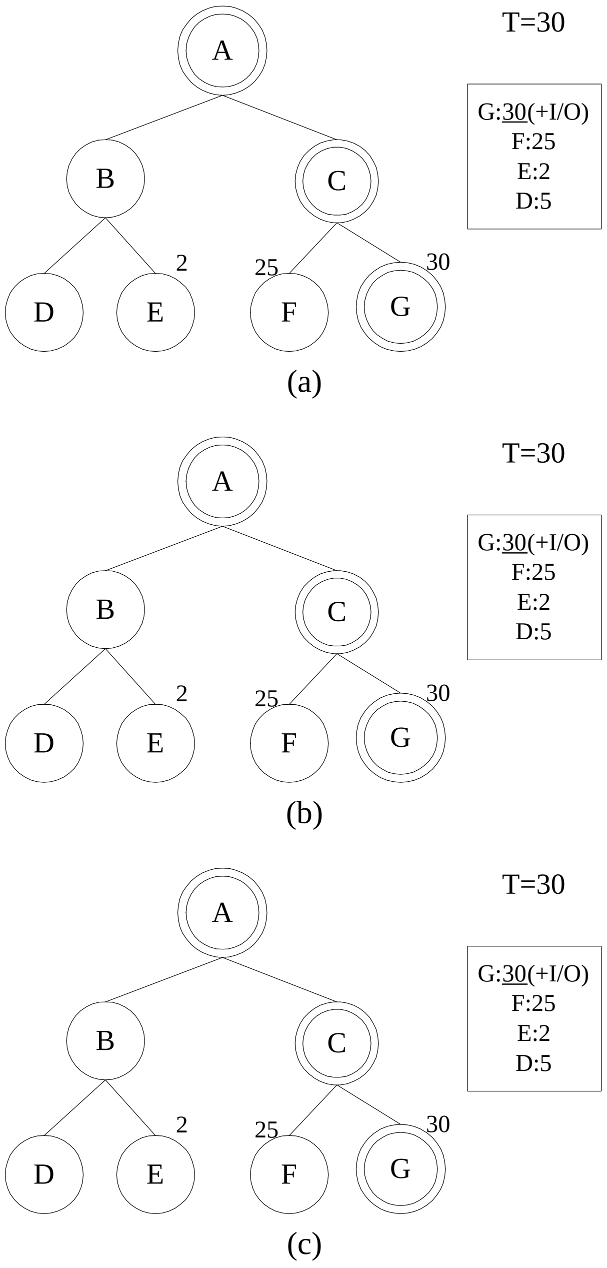

Fig. 6. A comparison on the P-gt pattern: (a) the Fast-Spread strategy with T = 30; (b) the Cascading strategy with T = 30; (c) the DT strategy with T_1 = 40, T_2 = 25, T_3 = 15.

III. The Dynamic DT Strategy

In this section, we present how to replicate data with different thresholds in Data Grids. First, we describe the details of our data structure. Next, we present the dynamic DT strategy.

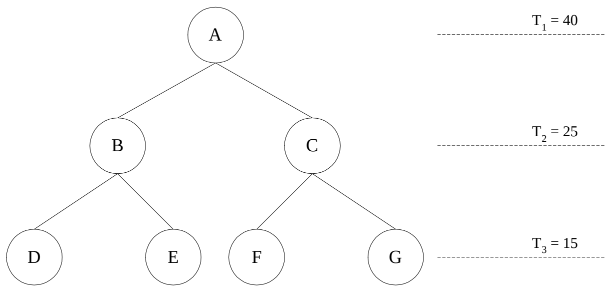

Fig. 7. Different thresholds of layers

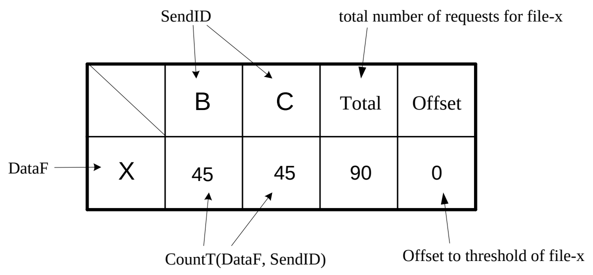
SendID
total number of requests for file-x
B
C
Total
Offset
DataF
X
45
45
90
0
CountT(DataF, SendID)
Offset to threshold of file-x

Fig. 8. The requesting record

A. Data Structure

In [2], Chang *et al.* assume that the clients consist of a binary tree structure. Once the root of the tree gets the request for a file, the root sends the file to the client. The tree structure of the grid implies that there is only one shortest path by which the messages and files can travel to get to their destination. Moreover, they set the different threshold T_i to the i-th layer of the tree structure, where $T_i > T_{i+1} > 0$, i is the i*-th* layer of the tree. We also assume that the layer of the root is 1, and the layer of the leaf node is h, where h is the height of the tree structure.

Figure 7 shows an example of the tree structure of the dynamic DT strategy. In this example, each node, except for the leaf nodes, keeps a record of requests for file $DataF$ when each file $DataF$ is requested from the child node $SendID$. We name the record *CountT*, and the related content is shown in Figure 8, where the $CountT(DataF, SendID)$ at the node $RecID$ means that the number of requests for $DataF$ from node $SendID$.

B. The Algorithm

Among large amount of different data files, there may exist some hot data files. That is, the difference of the number of requests for each data file may be very large. Those files which have been mostly requested are hot data files. In the static DT strategy, they set different threshold T_i at each layer i before replicating. In the dynamic DT strategy, besides the static thresholds at each layer, we set an offset to the threshold for each file, which may be adjusted while the replication of the file occurs. This allows data replication of hot files to occur earlier than for others by decreasing the thresholds of hot files earlier than the normal ones. We assume that each file can have its own threshold. If the number of requests for the file is more than half the number of total requests, the offset to the threshold of the file will be reduced.

```
procedure $HandleRequestF(SendID, RecID, DataF)$
begin
  if (data $DataF$ does not exist at node $RecID$) then
    begin
      Let $PID$ be $RecID$'s parent node;
      Send a RequestF(RecID, PID, DataF) message to $RecID$'s parent node $PID$;
      if (node $RecID$ is not a leaf node) then
          CountT(DataF, SendID) := CountT(DataF, SendID) + 1;
    end
  else /*file $DataF$ exists at node $RecID$*/
      ReplicateD(DataF, SendID, RecID);
end;
```

Fig. 9. Procedure $HandleRequestF$

The procedure $HandleRequestF$, as shown in Figure 9, handles the action when node $RecID$ receives a *RequestF* request message for data file $DataF$ from its child node $SendID$. If the requested file $DataF$ does not exist at node $RecID$, node $RecID$ will forward the request to node $RecID$'s parent node, and *CountT(DataF, SendID)* is increased by one. While the node $RecID$ having file $DataF$ receives a request from node $SendID$, procedure *ReplicateD* is executed, as shown in Figure 10.

The procedure $ReplicateD$ handles the action when node $RecID$ having file $DataF$ receives a request from node $SendID$. If the node $RecID$ is not a leaf node, it will check the record $CountT$

```
procedure $ReplicateD(DataF, SendID, RecID)$
begin
  if (node $RecID$ is not a leaf node) then
  begin
    check_record(DataF, SendID);
    Send a SendF(SendID, FileF, FlagR) message to $RecID$'s child node $SendID$;
  end;
end;
```

Fig. 10. Procedure $ReplicateD$

and send a message $SendF$ to $RecID$'s child node $SendID$.

Figure 8 shows the requesting record of the dynamic DT strategy. In the requesting record $CountT$, we record the number of requests for each data file and also count the summation. Moreover, we add a column $Offset$ for each data file. In the dynamic DT strategy, each data file $DataF$ has its own offset OTF_{DataF} to the threshold which belongs to its layer. Initially, we set the offset OTF_{DataF} = 0. While replicating, the node calculates the number of requests for the data file $DataF$. If the number of requests for the file is more than half the number of total requests, we assume that the file is hot, and decrease the offset OTF_{DataF} of the data file $DataF$ by α. As shown in Figure 11, we check the record $CountT$ to find the number of previous requests from node $SendID$. If the count of *CountT(DataF, SendID)* is larger than the summation of the threshold T_{SendID_Layer} and the offset OTF_{DataF}, the node will calculate the number NR_{DataF}, the summation of the number of requests for the data file $DataF$, and TNR, the total number of all requests at node $RecID$. Then, it compares NR_{DataF} with TNR. If NR_{DataF} is more than half of TNR, the offset OTF_{DataF} of the data file $DataF$ will be decreased by α, where α is a constant. Then, we transfer the replica of the data file with the offset OTF_{DataF}. The child node which receives the offset OTF_{DataF} updates the offset of the file on its own record.

We illustrate the dynamic DT strategy in Figure 12. First, as shown in Figure 12–(a), after node G requests file X, file X is replicated at node A. We assume that node A already has file X, and the

```
procedure $check\_record(DataF, SendID)$
begin
  if (CountT(DataF, SendID) $\geq$ ($T_{ChildLayer}$ + $OTF_{DataF}$)) then
  begin
    $NR_{DataF}$ := the summation of the number of requests for the data file;
    $TNR$ := the total number of all requests at node $RecID$;
    if ($NR_{DataF} \geq \frac{1}{2}\, TNR$)then
      $OTF_{DataF}$ = $OTF_{DataF}$ - $\alpha$;
    Let $CountT(DataF,\ Offset_{DataF})$ be the $OTF_{DataF}$ from its parent;
    CountT(DataF, SendID) := CountT(DataF, SendID) - $T_{ChildLayer}$;
    $FlagR$ := 1; /*A flag to indicate whether $DataF$ should be replicated*/
  end;
end;
```

Fig. 11. Procedure $check\_record(DataF, SendID)$

offset of file X is -10. Then, the requests for the data files are continually generated from the client nodes. In Figure 12–(b), the number of requests on the records of node A and B reach each own threshold. While node A receives the request for file X from node E through node B, node A checks the record $CountT$ to find the number of previous requests for file X. It will compare *CountT(X, B)* with the threshold T_2 and then check whether the summation of the number of requests for file X, such as 95, is more than half of the total number of all requests for file X at node A or not, and if yes, the offset OTF_{DataF} of file X will be decreased by a constant α, where α is 10. Now the count of requests for file X reaches the threshold T_2, and file X will be replicated at node B. Then, node A will send file X to node A and update the offset of file X. Finally, as shown in Figure 12–(c), the replica of file X is replicated at node B and node E, and the offsets of file X at node B and node E are updated by -20.

IV. Performance

In this section, we compare the performance of our dynamic DT strategy with the static DT strategy

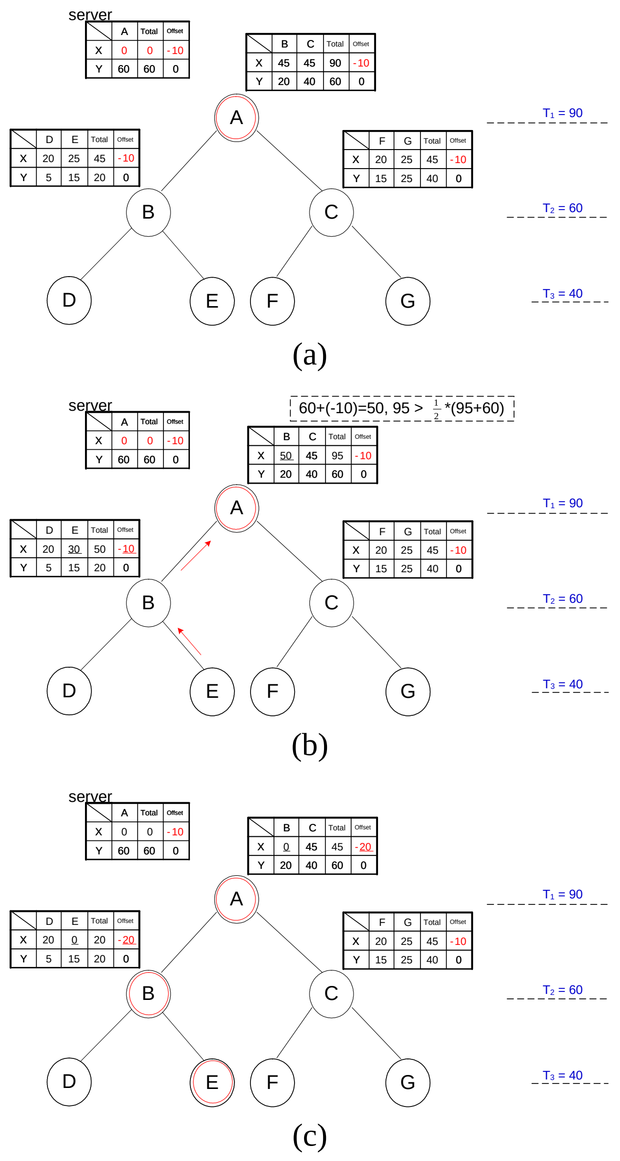

Fig. 12. An example of the dynamic DT strategy: (a) file X is replicated at node A; (b) the number of requests on the records of node A and B reach each own threshold; (c) the file X is replicated at nodes B and E, and the offsets are changed.

[2], the $Fast$-$Spread$ strategy, and the $Cascading$ strategy [10].

A. The Performance Model

In our simulation study for the distributed environment, we use the simulator $SimpackJ$ [8]. $SimPackJ/S$ is the $JavaScript$ and $Java$ version of $SimPack$, which means that $SimPackJ/S$ is a collection of $JavaScript$ and $Java$ libraries and executable programs for computer simulations. $SimPack$ is a general simulation toolkit for creating discrete event or continuous simulations. It is event-oriented, and its key point is that it supports a wide variety of event scheduling and continuous-time simulation models [4]. Our strategy is carried out based on the three-tier Data Grid topology [6].

We generate the requests for the files from the client nodes according to the $Poisson$ distribution with a parameter R_{AP}. That is, $\frac{1}{exp(R_{AP},N)}$ is the number of requests per second, where exp is an exponential distribution function. The parameter R_{AP} is the rate of requests, and N is the number of nodes. We specify different types of nodes: client, server, and cache nodes, which are described as follows.

- Server Node: It represents the main storage sites, where all or some parts of the data are stored. The sites represent the root of the grid hierarchy.
- Cache Node: It represents an intermediate storage site, for example, a regional storage site. Such sites would have high storage capacity and would replicate part of the data stored at the main storage site.
- Client Node: It represents the sites where data access requests originate and are generated. The client nodes are always placed at the leaf layer of the Grid hierarchy.

In our simulation study, the number of server node is 1, the client nodes are all in the lowest layer, and the cache nodes can be in several layers. Because the topology is a binary tree, the total number of sites can be calculated as $2^0+2^1+2^2+...+2^i$, where i is the number of layers. So, in our simulation, the total number of nodes is $2^0+2^1+2^2+2^3 = 15$.

Table I shows the parameters used in our strategy. N means the number of nodes. Due to the property of the binary tree, N is fixed, if the number of layers of the tree is chosen. NAP means the total number of access patterns in the system. $DataF$ means the file which is requested. T_i is the threshold of i-th layer, and it will be set at the beginning of the simulation.

Initially, all files are placed at the Main Storage Site (root). Moreover, we set the different threshold T_i at the i-th layer of the tree structure. On each node, except for the leaf nodes, there is a record of requests for some $DataF$ files, which each record is requested from its child nodes. The offset of each file will be decreased by a constant α, where α is

TABLE I
THE SIMULATION PARAMETERS

Parameter	Description
N	The number of nodes
NAP	The total number of access patterns
$DataF$	The data file
T_i	The threshold of each layer i
L_{AP}	Probabilities of the locality of access patterns
α	The constant used to reduce the offset

10. Two types of access patterns are considered in our simulation study, the random and the localized access patterns. The random access patterns are generated at the client nodes randomly. The locality of access patterns means that the files recently accessed are likely to be accessed again, or the files recently accessed by a client are likely to be accessed by nearby clients.

B. Simulation Results

In this section, we compare our DT strategy with the $Cascading$ strategy and the $Fast$-$Spread$ strategy [10]. Here, the access costs in Data Grids which we use are response time and the bandwidth consumption.

Given that the number of files are 100, we take measurements for two different access patterns, random and localized. We assume that the data which is initially stored in the server has the same size of 2 gigabytes each. The clients which all requests come from are assigned to be 8. Our strategy is carried out based on the three-tier Data Grid topology [6]. Then, we calculate the response time and bandwidth consumption of these three strategies, DT, $Cascading$, and $Fast$-$Spread$. In Figure 13 and Figure 14, we set T_1 = 160, T_2 = 80, and T_3 = 40 in DT, and T = 60 in the $Cascading$ strategy and the $Fast$-$Spread$ strategy.

Because the amount of data in the actual Data Grids is in the order of petabytes, to simulate such large data values, a scale of 1 file to 10,000 files is used [10]. Therefore, the storage capacity at each layer is also reduced. Table II shows the result of scaling.

Figure 13 shows the response time (in terms of the number of requests and transfers) of our DT strategy and the $Fast$-$Spread$ strategy under the case of the random access patterns. In the $Fast$-

TABLE II
THE SCALE OF DATA GRID ENVIRONMENT PARAMETERS

	Actual Size	After Scaling
Number of files	1,000,000	100
Storage at layer 1	2200 Terabytes	220 Gigabytes
Storage at layer 2	1000 Terabytes	100 Gigabytes
Storage at layer 3	120 Terabytes	12 Gigabytes

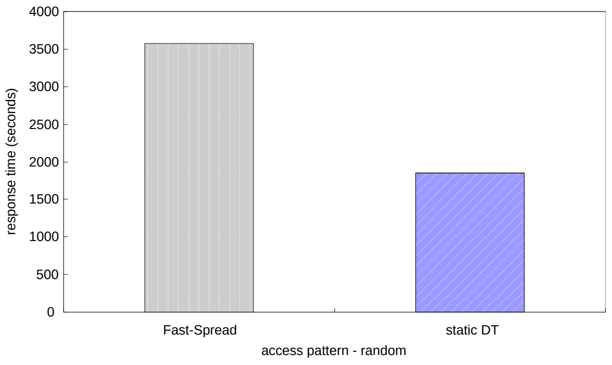

Fig. 13. A comparison of the response time between the $Fast$-$Spread$ strategy and the DT strategy

$Spread$ strategy, there is only one threshold, and the record of the number of requests for a file is stored at the client which the requests initially come from. Hence, when the number of requests reaches the threshold, the file is replicated along the path to the client. In our DT strategy, we set a different threshold at each layer. When the number of requests reaches the threshold of some layer, the file is replicated to the cache node or the client of that layer. From Figure 13, we show that our strategy needs the less response time than the $Fast$-$Spread$ strategy.

Figure 14 shows the response time of our strategy and the $Cascading$ strategy based on the local access patterns. In the $Cascading$ strategy, there is only one threshold. Hence, when the number of requests reaches the threshold, the file is replicated to the cache node which is below the server and on the path to the client. From Figure 13, we show that our strategy needs the less response time than the $Cascading$ strategy.

Next, we compare bandwidth consumption in our strategy with that of the previous strategies, as shown in Figure 15. In this case, we take measurements of the random access patterns. When a node requests a file, and then the replica of the

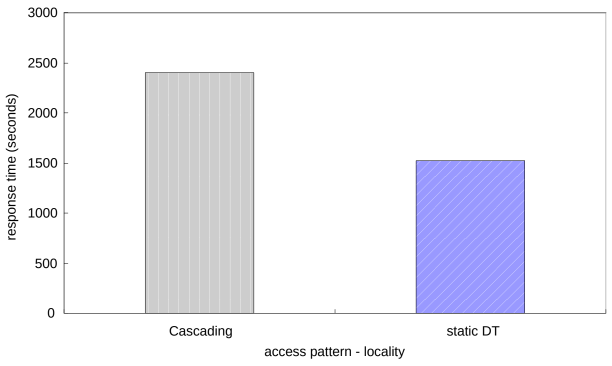

Fig. 14. A comparison of the response time between the $Cascading$ strategy and the DT strategy

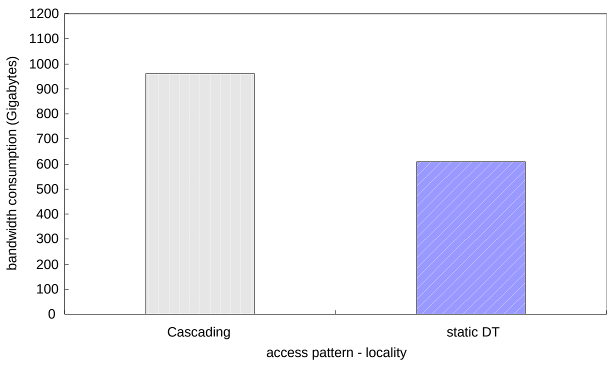

Fig. 16. A comparison of the bandwidth consumption between the $Cascading$ strategy and the DT strategy

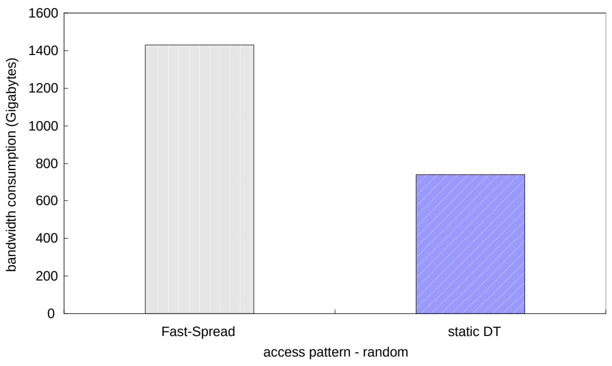

Fig. 15. A comparison of the bandwidth consumption between the $Fast\text{-}Spread$ strategy and the DT strategy

file is replicated and transferred to another node, we assume that the occurrence of the bandwidth consumption is the amounts of the transference of data. Figure 16 shows the bandwidth consumption of our strategy and the previous strategies. In this case, we take measurements of the locality access patterns.

V. CONCLUSION

In this paper, we have proposed a dynamic different threshold strategy for data replication in Data Grids to reduce the number of requests for the hot files. In the dynamic DT strategy, each data file has its own threshold which may be adjusted while the replication of the file is occurring. This allows data replication of hot files to occur earlier than the replication of others by setting an offset to the threshold of each file and decreasing the thresholds of hot files earlier than the normal ones. In the simulation study, we have shown that the performance of the dynamic DT strategy is better than that of the static DT strategy. How to handle the case of the movement of the code of application program that is one possible future research direction.

VI. ACKNOWLEDGEMENT

The authors also like to thank “Aim for Top University Plan” project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

REFERENCES

[1] R. S. Chang, J. S. Chang, and S. Y. Lin, “Job Scheduling and Data Replication on Data Grids,” *Future Generation Computer System*, Vol. 23, No. 7, pp. 846–860, 2007.
[2] Y. I. Chang, L. W. Huang, W. H. Yeh, and Y. W. Huang, “A Different Threshold Strategy to Data Replication in Data Grids,” *Proc. of the 4th Workshop on Grid Technologies and Applications,* pp. 21–28, 2007.
[3] G. Coulouris, J. Dollimore, and T. Kindberg, “Distributed Systems, concepts and designs,” *third Edition, Addisson Wesley*, 2001.
[4] P. A. Fishwick, “SimPack: getting started with simulation programming in C and C++,” *Proc. of the 24th Conf. on Winter Simulation,* pp. 154–162, 1992.
[5] H. Jin, J. Huang, X. Xie, and Q. Zhang, “Using Classifi cation Techniques to Improve Replica Selection in Data Grid,” *OTM Conferences,* pp. 1376–1387, 2006.
[6] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman, “Simulation of Dynamic Data Replication Strategies in Data Grids,” *Proc. of the 17th Int. Symp. on Parallel and Distributed Processing,* pp. 1–10, 2003.
[7] J. Nabrzyski, J. M. Schopf, and J. Weglarz, “Grid Resource Management: State of the Art and Future Trends,” *Springer Publishers*, 2003.
[8] M. Park, and P. A. Fishwick, “SimPackJ/S: A Web-Oriented Toolkit for Discrete Event Simulation,” *Proc. of SPIE Vol. 4716,* pp. 348–358, 2002.

[9] K. Ranganathan, and I. Foster, "Design and Evaluation of Dynamic Replication Strategies for a High-Performance Data Grid," *Proc. of the Int. Conf. on Computing in High Energy and Nuclear Physics,* pp. 106–118, 2001.

[10] K. Ranganathan, and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," *Proc. of the Int. Workshop on Grid Computing,* pp. 75–86, 2002.

[11] Y. Saito, and H. Levy, "Optimistic Replication for Internet Data Services," *In Proc. of the 14th Int. Conf. on Distributed Computing*, pp. 297–314, 2000.

[12] J. Smith, and M. Jones, "Survey and Taxonomy of Grid Resource Management Systems," *Chaitanya Kandagatla University*, 2003.

[13] M. Tang, B. S. Lee, C. K. Yeo and X. Tang, "Dynamic Replication Algorithms for the Multi-tier Data Grid," *Future Generation Computer System*, Vol. 21, No. 5, pp. 775–790, 2005.