A Predictive Algorithm for Replication Optimization in Data Grids

Ruay-Shiung Chang, Ning-Yuan Huang and Jih-Sheng Chang
Network Innovation Technology Laboratory
Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien, TAIWAN, R.O.C
rschang@mail.ndhu.edu.tw

Abstract

Previous replication algorithms in grids only consider the file access times. They do not put user access patterns into consideration. In this paper, we propose a predictive algorithm for replication based on user's behaviors. The basic assumption is that if a user accesses a file, then he will access this file again. We can predict what the next file would be accessed with largest probability according to the access history. Our algorithm is modified from the context modeling algorithm proposed by Thomas M. Kroeger in 2001. We adapt his algorithm for Grid environment.

1. Introduction

Replica management is an important issue in a Data Grid System [3]. The main goals of data replication are to improve data access latency and fault tolerance. Well deployed data replication can also help in load balance and increase reliability by creating multiple copies of the same data file in a Data Grid.

The replication is usually done after a request is arrived. Upon-request replication will cause a significant delay. If we can perform replication in advance before a file is really requested, we can reduce a significant delay. For example, if node A requests a file F residing on node B, node A has to suspend the current work and fetch F from B. When file fetching is complete, the suspended work can be resumed. In this paper, we use a prediction based method to predict which file should be replicated. We also use a cost estimation algorithm to find a node for replication to reduce the average access cost.

User's behavior is predictable. If somebody used a file in the past, he would like to get the same file in the future possibly. According to user's access patterns, we can do some prediction. By analysis and data mining, we can predict which files a user wants and pre-fetch them to the site beforehand.

Most of previous studies [8, 9] treat each user access as an independent event. They don't care how important the sequence of file accesses is. The result is they lose a lot of useful information. Previous replication algorithms are based on the assumption that popular files of one site are also popular at another site. A site counts the number of request for each replica. According to the model, the best replica will be one that is the most popular replica. Though this is a simple heuristic, it does not consider contextual information. The replication algorithm proposed in this paper is based on data popularity with consideration for spatial and temporal locality.

Predictive pre-fetching method is based on the hypothesis that previous access patterns provide information for the future accesses. In a Grid system, the jobs submitted by the same user may have similar data access patterns. For example, a physicist needs to do some physical analyses. He/She develops data analyzing codes in order to discover physical parameters from the data. In many cases, the result of one job is probably the input of another job. As a result, a specific set of data files will be accessed frequently. For another example, analyses are not only executed in the context by one single user, but are also shared within a given analysis group. A user usually accesses data that is generated by another user within the same analysis group. These two examples illustrate how a file access pattern can provide information about future accesses.

By using data mining techniques, we can find out the access pattern. The access pattern is about the relation between a user and the file that has been accessed. If we discover the relation between them, we can know which file would be accessed next with the highest probability.

2. Related Work

In [9], Ranganathan and Forster proposed and evaluated the following results:

If the access pattern is fully random, fast spread is the best strategy.

A. If the access pattern has sufficient locality, cascading is the best choice.
B. If we concern about bandwidth consumption, fast spread works best.

The conclusions tell us that access pattern is an important factor for replication strategy design. An access pattern for a Grid user and a Grid job is a critical information.

A data access optimization process is proposed in [8].

László Csaba Lőrincz et al. proposed an analysis process by the following steps:

A. Monitor the execution of jobs and get the resource access information.
B. Analyze the information and generate the behavior descriptions.
C. Implement an optimized replication algorithm according to the behavior descriptions.

Though they proposed an analysis process, they did not provide any actual methodology that can achieve this goal.

In [10], the authors proposed a data-mining based replication strategy. They use a technique called the K-Nearest Neighbor (KNN) rules to select the best replica locally. For example, assume each data sample has n attributes, the distance between two data sample $P = \langle p_1, p_2, \cdots, p_n \rangle$ and $Q = \langle q_1, q_2, \cdots, q_n \rangle$ is defined as the following equation:

$$d(P,Q) = \sqrt{\sum_{n=1}^{n} (p_i - q_i)^2}$$

It finds the best site from the previous history of file transfers. If there are sufficient access logs in a storage element, it can find the best site by the KNN rule. Otherwise, it uses the traditional method. The KNN based replication strategy focuses on the site which can accelerate transfer when replicas are accessed. It doesn't have predictive ability to pre-fetch files. All replications are done after a request is arrived. In this paper, we provide a predictive method to pre-fetch files before the file requests really come.

In [7], Thomas M. Kroeger and Darrell D. E. Long proposed a method called Extended Partial Context Modeling (EPCM). EPCM is originated from Partitioned Context Modeling (PCM) [5] and Finite Multi-Order Context Modeling (FMOC). FMOC is based on the partial pattern matching, a text compression algorithm [1]. We can predict by using the multiple order context. This technique is called FMOC. A trie, or prefix tree, is an ordered tree that is used to store an associative array where the keys are strings. Each file access is modeled as a symbol (an alphabet or the file name). We can use this information for predicting the next file.

PCM divides the trie into several partitions. Each partition contains first order node and its entire descendants. The number of nodes in a partition is limited to a constant number. If the number of nodes in a partition reaches the limitation, we say that partition is full. When we add a new node into a partition that is full, all node counts in the partition are divided by two and rounded to the nearest integer. Any node with zero count will be cleared to make space for new nodes. If there is no space available, the new node adding process is ignored.

PCM can be further modified to achieve more accurate prediction. The modified PCM is called EPCM. EPCM predicts not only one node but also all the descendant with the probability higher than a threshold.

3. Grid Access Pattern Modeling

We have proposed a predictive pre-fetch mechanism called Grid Access Pattern Modeling (GAPM).The GAPM has three phases. They are monitor stage, analyzing stage and replication stage. Software agents are placed in each storage element. In the monitor phase, an agent in each node sniffs regular replication operations and other file access events. After getting the data reports by an agent, the analyzer keeps tracking and does data mining on them. The analyzer figures out each replica access pattern. When a job accesses some files, the replicator pre-fetches the most possible replica that is needed to that site.

The components in this model are described as the followings: Users submit jobs to a job scheduler via an interface provided by a job broker. Users are distinguished by their identifications. Each user has favorite or preferred jobs or applications. These jobs or applications may access the same set of files. We can predict future file accesses according to user's behaviors. A job scheduler assigns jobs to Grid sites. In a Grid site, a computing element is the resource that provides computing power for users. A resource having computing power means that it can execute programs or applications. A storage element is the resource that provides storage capacity for users. A storage element is modeled as a finite amount of storage space. A replica catalog provides a query interface for the mapping from a logical file name to its corresponding physical file names. A physical file name includes information about where a replica stores. A replica optimizer plays an important role in this model. It finds which replica will be accessed most likely. And then it replicates files with the highest access probability to a site that a job is assigned in. The GPAM analyzer is deployed in a replica optimizer. It is a centralized predictive method. A centralized method has one advantage: it makes a decision based on all information.

A mining engine maintains a trie. A trie is a data structure based on a tree, but it's more efficient for storing previously seen access patterns and maintaining the count of each pattern. Figure 1 shows an example trie for GAPM. It has a common root. The nodes in second level use user name or identification as keys. The nodes in third level and lower levels use file name as keys. By the trie, we can track different users' access patterns. There are some differences between different user's access patterns. So we use individual trie to track their access patterns. The path to each node form its ancestor node in the second level represents a file access sequence. The length of a path is called an order. Each node in the third and lower levels has a count. This count is used to record the number of times this sequence has occurred. Because memory capacity is finite, there is a parameter called track order to determinate how long we track file access sequences. The best value of track order will be described later. The algorithm modified from [6] for adding access events is shown as Figure 2.

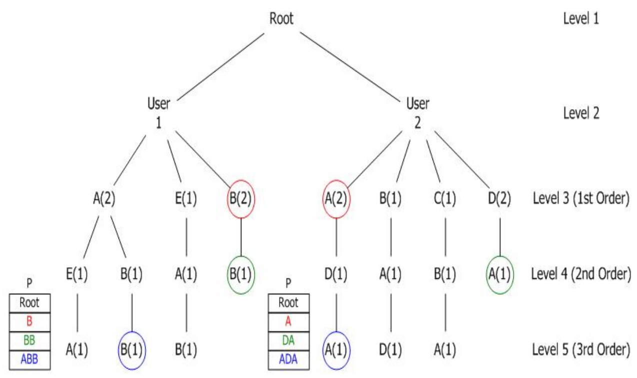

Figure 1. An example trie of GAPM

```
PCM-Insert(T,s) // We want to add access event s into trie T
1  P[0..m]=P of T // P is an array of pointer, P[0] = root(T)
2  For i ← m-1 to 0 {
3      if P[i] ≠ NULL then {
4          if P[i] has a child with key s then {
5              add the count of the child with key s by 1
6              set P[i+1] to point to the child with key s
7          } else {
8              if the partition which we want to add node is full then {
10             Every node in this partition divided the count
                   by 2 and removed the node with count
                   lower then 1
11             if the partition that we want to add node is still full then {
12                 abondon adding node
13             }
13         } else {
14             add a child with key s to the node pointed by P[i]
15             set count of the child with key s to 1
16         }
17     }
18 }}
```

Figure 2. PCM insertion algorithm

For our GAPM trie, each user has a subtree rooted at the second level. The algorithm for inserting an access event is shown below.

```
GAPM-Insert(T, u, s) // T is a trie, u is a user identification, s is an access event
1  R ← root(T)
2  If R does not has a child with key u then {
3      Add a child with key u
4  }
5  Let U is a subtrie that contains the child with key u and its entire descendants
6  PCM-Insert(U,s)
```

Figure 3. GAPM insertion algorithm

By the algorithms described above, we can construct a trie like Figure 1.

With the GAPM trie, we describe how to get the predictive information such as the file which is the most likely that will be accessed next. We can perform some replications for files if their probabilities are greater than a pre-set threshold. The best threshold will be estimated in our simulations.

By our model, we can predict access patterns for different users. When a job is executed in a computing element, this computing element maintains a prediction pointer array P. Each time a job runs in a computing element, an empty pointer array P will be initialed. It records the pointer to trie nodes. But it is different from the array that is used during the construction process. We use a different pointer array to store the information about where we traversed. The traversal procedure and the insertion procedures are similar during searching child and operating pointer array. But, we traverse the trie without any change. Because prediction is based on partial sequence, so we don't change the trie during prediction. If we change the trie during the prediction, it may cause multiple sequences from different jobs to mix up.

There is another parameter which is called prediction order in our model. The prediction order *o* means that we predict the next file according to a sequence of length *o.* We use the predict point array P. Assume a subtrie *S* that contains a root node is pointed by P[*o*] and all its descendants. We use the following formula to calculate the probability of each descendent occurring during the next event. In this formula, the $count_{child}$ is the count of a node other then the root node in this subtrie. The $count_{parent}$ is the count of root in this subtrie *S*.

$$probability = \frac{count_{child}}{count_{parent}}$$

We can pre-fetch files that have their probability greater then a pre-defined threshold. This threshold is a parameter we set in this model.

We can use the trie predicting algorithm to perform prediction for a user. In Figire 4, *T* is a subtrie that contains its root node and its all descendants. The root node uses the user's identifier as key.

```
Trie-Predict(o, T, t, s) // o is the prediction order, T is the
subtrie, t is the pre-fetch threshold, s is an event we see
now
1  Assume P[0..m] is a pointer array. It is empty when we
   predict first time for a job. Otherwise, it stores previous
   state for the same job.
2  L ← Empty List //L is a list for return predictive
information
3  For i ← m-1 to 0 {
4      if P[i] ≠ NULL and P[i] has a child with key s then
{
5          set P[i+1] to point to the child with key s
6      }
7  }
8  for each descendant D of the node pointed by P[o] {
9      cc ← count of D
10 cp ← count of the node that is pointed by P[o]
11 if cc/cp is greater than t then {
12     add D to L
13 }
14 }
15 return L
```

Figure 4. Trie predict algorithm

We use the algorithm GAPM-Predict (Figure 5) for selecting the subtrie.

```
GAPM-Predict(o, T, t, u, s) // o is the predict order, T is a
trie, t is the pre-fetch threshold, u is a user identification,
s is a event we see now
1  R ← root(T)
2  If R does not has a child with key u then {
3      return nothing
4  }
5  return Trie-Predict(o, T, t, s)
```

Figure 5. GAPM predict algorithm

Our algorithm can't predict under the following two conditions: first, if we don't know any information about access pattern of somebody, we can't predict the next file he will access; second, if the node of some order we want to predict has no descendents, we can't predict anything. Under these two conditions, GAPM predicting algorithm returns an empty list.

If the probabilities of some files are larger than a pre-defined threshold, the replicator will replicate the file to the place that the job runs.

4. Simulations and Performance Analysis

We use OptorSim [2] to evaluate the performance of our predicting model. OptorSim is developed according to the real data Grid system in Europe. OptorSim is a project of EDG (EU Data Grid) and is a simulation toolkit based on Java technology.

There are eighteen sites, some of them are routers, in our simulation environment. A router is a site without computing elements and storage elements. The site 8 is allocated a storage element to hold all of the master files but without any computing elements. The site 8 has 10 Tera Bytes of storage capacity. The other sites have 50 Giga Bytes of storage capacity.

Table 1 specifies the simulation parameters in our study. Each site has a limited storage capacity. There are six users, and there is no overlap between each user's accessed file set. Each user has their job types. Each set of files is assumed to be composed as 10GB files. The users submit jobs at a regular time interval until all jobs are done.

Table 1. Simulation parameters

Parameter	Vaule
Number of Sites	11 (without routers)
Storage Space in Each Site	50GB
Number of Jobs	1000
Number of users	6
Size of Single File	1GB
Total number of files	97
Number of Simulations	100

There are three parameters in our model: pre-fetch threshold, track order and prediction order. We intend to find out what value for these parameters would offer the best performance. We use a hit ratio to measure the performance. The hit ratio is calculated by the following equation. Assume *J* is a set of all jobs, K_j denotes the set of files that are accessed by job *j*. *k* is an accessed file belongs to K_j. If *k* is pre-fetched to the local storage element when job *j* requires it, then the value of $\delta_{j,k}$ equals to 1. Otherwise, $\delta_{j,k}$ equals to 0. Then define the hit ratio r as:

$$r = \frac{\sum_{j \in J, k \in K_j} \delta_{j,k}}{\sum_{j \in J} \left| K_j \right|}$$

Figure 6 shows that average number of pre-fetched files decreases significantly while increasing the pre-fetch threshold. Pre-fetching too many files may fill up our storage elements. Each value is a result from 1000 jobs executions. The pre-fetch threshold is set in range from 0 to 0.975 with an interval of 0.25. GAPM track order is set to 4 and the prediction order is set to 2. We want to determine the appropriate value for high hit ratio with the least overhead. When the pre-fetch threshold is larger than 0.5, the average number of pre-fetched files in intended to be more stable. According to the results from Figure 6, we can be sure that the best value is 0.5 with high hit ratio and little transfer overhead. So we choose 0.5 as our pre-fetch threshold.

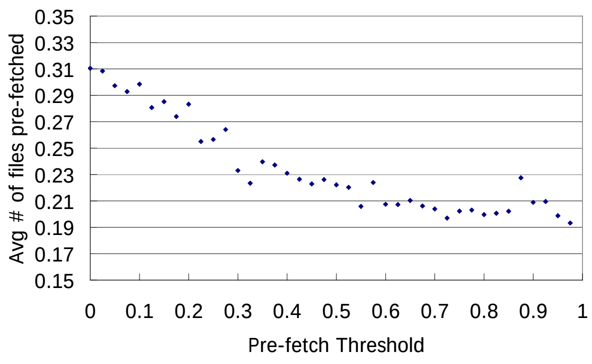

Figure 6. Average number of pre-fetched file for every file request

With the determination of the best setting of threshold, we have to discover the best value of prediction order and track order in our GAPM algorithm. Figure 7 shows the average number of pre-fetched files under different prediction order and track order. Each value is extracted from the result of 1000 job executions. Figure 7 shows the relation between the average number of pre-fetched files and the hit ratio. When average number of pre-fetched files is 0.22 (the point in the circle), the hit ratio can reach a relative high value of 0.88. Increasing the number of pre-fetched files, the hit ratio does not increase noticeably. Accordingly, we set track order as 4 and prediction order as 2.

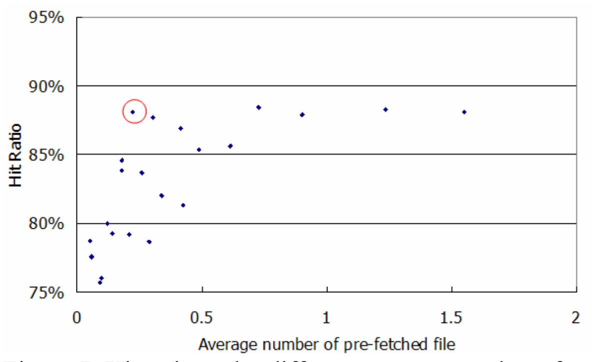

Figure 7. Hit ratio under different average number of pre-fetched files

We have evaluated the local storage hit ratio as shown in Figure 8. Compared with KNN rule, Cache only algorithm and unmodified EPCM algorithm, our algorithm has a higher hit ratio, because our algorithm pre-fetches files according to the access patterns of users.

The average reduced file processing time compared to plain replication upon request is presented in Figure 9. It shows how much the file processing time is saved compared to the cache only algorithm. Our GAPM predictive algorithm outperforms the other three mechanisms by considering user's access patterns and pre-fetching files that may be required. The simulation results show that our algorithm is more suitable than the other mechanisms for a Grid system.

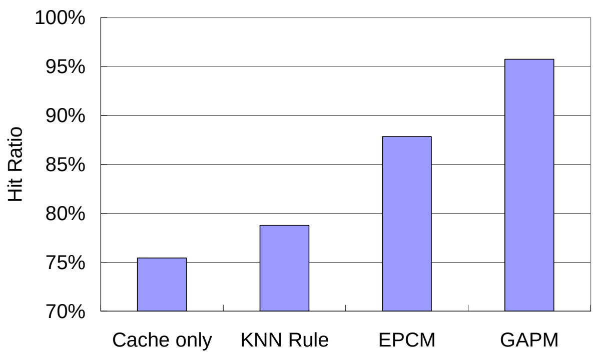

Figure 8. Local storage hit ratio

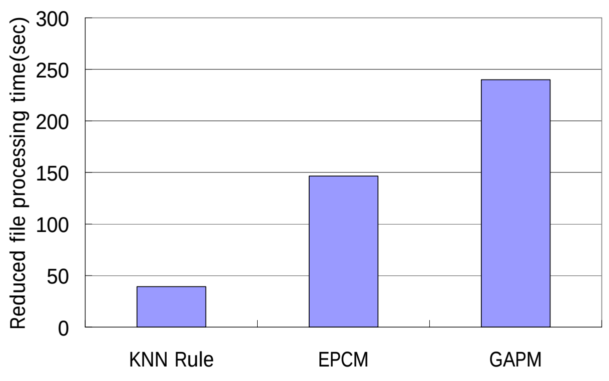

Figure 9. Average of reduced file processing time

5. Conclusions and Future Work

In this paper, we have addressed the problem of data movement in Data Grid environment. We focus on user's access pattern, which is an important factor that has never been studied so far. To use this useful information, we have proposed a user oriented prediction algorithm (GAPM). It can reduce job execution time by pre-fetching files that are required.

We have evaluated the efficiency of our prediction algorithm by a Grid simulator called OptorSim. We use the hit ratio and average file transfer time as the benchmark. We study and evaluate performance of other two different replication strategies. According to our simulation results, our predictive algorithm is superior to others in average job execution time.

In the future, the dynamic pre-fetch threshold and track/prediction orders in our paper can be a further research direction. Other data mining techniques may be used in this field, such as neuron network and other statistical approaches [4]. In our approach, the prediction is based on ordered access sequences. It is possible that unordered access records can provide more predicting accuracy. Furthermore, when the system we want to deploy our prediction algorithm has too many users, our algorithm may spend a lot of space. The data structure has to be modified to adopt this kind of environment. In addition, we will implement our algorithm into Taiwan UniGrid project [12] and DAGS project [11] to realize our algorithm in real Grid systems.

6. References

[1] Timothy C. Bell, John G. Cleary, Ian H. Witten, *Text Compression*, Prentice Hall, Englewood Clis, New Jersey, 1990.

[2] William H. Bell, David G. Cameron , Luigi Capozza , A. Paul Millar , Kurt Stockinger , Floriano Zini, "Simulation of Dynamic Grid Replication Strategies in OptorSim," *Proceedings of the Third ACM/IEEE International Workshop on Grid Computing(GRID2002)*, Baltimore, USA, vol. 2536 of Lecture Notes in Computer Science, Pages: 46 - 57, November 18, 2002.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, Pages:187-200, 2001

[4] Jiawei Han, Micheline Kamber, Data Mining: Concepts and Techniques, New York, Morgan Kaufmann Publishers, Pages: 279–326, ISBN 1-55860-489-8, August 2000.

[5] Thomas M. Kroeger, Darrell D. E. Long, "Predicting File System Actions from Prior Events," *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, USA, Pages 319-328, 22–26 January 1996

[6] Thomas M. Kroeger, Darrell D. E. Long, "The case for efficient file access pattern modeling," *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, USA, Pages:14-19, March 1999

[7] Thomas M. Kroeger, Darrell D. E. Long, "Design and Implementation of a Predictive File Prefetching Algorithm," *Proceedings of the 2001 USENIX Annual Technical Conference*, Boston, USA, 25–30 June 2001

[8] László Csaba Lőrincz, Tamás Kozsik, Attila Ulbert and Zoltán Horváth, "Data access optimization on grid systems," *Proceeding of 14th IEEE International Workshops on Enabling Technologies*, Linköping university, Sweden, Pages:319 – 324, 13-15 June 2005

[9] K. Ranganathan, I. Foster, "Design and evaluation of dynamic replication strategies for a high performance data Grid," *Proceeding of International Conference on Computing in High Energy and Nuclear Physics*, Beijing, P.R.China, 3-7 September 2001

[10] Rashedur M. Rahman, Ken Barker, Reda Alhajj, "Replica selection in grid environment: a data-mining approach," *Proceedings of the 2005 ACM symposium on Applied computing*, Santa Fe, USA, Pages: 695 – 700, 2005

[11] Digital Archive on Grid System website, http://portal.csie.ndhu.edu.tw/dags/

[12] Taiwan UniGrid Project Portal website, http://unigrid.nchc.org.tw/