Optimal Placement of Replicas in Data Grid Environments with Locality Assurance

Yi-Fang Lin Pangfeng Liu
Department of Computer Science
National Taiwan University
Taipei, Taiwan, R.O.C.
pangfeng@csie.ntu.edu.tw

Jan-Jan Wu
Institute of Information Science
Academia Sinica
Taipei, Taiwan, R.O.C.

Abstract

Data replications is a typical strategy for increasing access performance and data availability in Data Grid systems. Current work on data replication in Grid systems focuses on infrastructure for replication and mechanisms for creating/deleting replicas. The important problem of choosing suitable locations for placing replicas in Data Grids has not been well studied.

In this paper, we address the problem of data replica placement in Data Grids given the traffic pattern and locality requirements. We propose a new placement algorithm that finds the optimal locations for the replicas so that the workload among these replicas is balanced. We also propose a new algorithm to decide the minimum number of replicas required when the maximum workload capacity of each replica server is known. All these algorithms ensure that locality requirements from the users are satisfied.

1 Introduction

Grid computing is an important mechanism for utilizing distributed computing resources. These resources are distributed in different geographical locations, but are organized to provide an integrated service. A grid system can provide computing resources so that users at different locations can utilize the CPU cycles of remote sites. In addition, users can access important data that are available only in several locations, without the overheads of replicating them locally. These services are provided by an integrated grid service platform so that user can access the resource transparently and effectively.

One class of grid computing and the focus of this paper is Data Grids that provide geographically distributed storage resources to large computational problems that require evaluating and managing large amount of data [3, 11, 16]. For example, the scientists working on bioinformatics may need to access human gnome databases on different remote locations. These databases have tremendous amount of data, so the cost of maintaining a local copy on each site that needs the data is extremely expensive. In addition, these databases are mostly read-only, since they are the input data to the applications for various purposes, such as benchmarking, identification, and classification. With the high latency of wide-area network that underlies most Grid systems, and the need to access/manage several petabytes of data in Grid environments, data availability and access optimization becomes key challenges to be addressed.

An important technique to speed up data access for Data Grid systems is to replicate the data in multiple locations, so that a user can access the data from a site in his vicinity. It has been shown that data replication not only reduces access costs, but also increase data availability in many applications [11, 17, 15]. There is a fair amount of work on data replication in Grid environments. However, most of the existing work focused on infrastructures for replication and mechanisms for creating/deleting replicas [4, 7, 6, 8, 11, 15, 18, 17, 19]. We believe that, in order to obtain maximum gains of replication, a strategic placement of the replicas is necessary.

A number of early works address placement of data replicas in parallel and distributed systems with regular network topologies such as hypercubes, torus, rings, and trees. These networks posses many attractive mathematical properties that enable the design of simple and robust placement algorithms [2, 12, 21]. These algorithms, however, cannot be directly applied to Data Grid systems due to hierarchical net-

work structures and special data access patterns in Data Grid systems that are not common in traditional parallel systems. An initial work on replica placement for Data Grids was reported in [1]. The author proposed a heuristic algorithm, named `Proportional Share Replication`, for the placement problem. However, the algorithm does not guarantee to find the optimal solution.

In this paper, we study replica placement in Data Grid systems, taking into account several important issues described below. First, the replicas should be placed in proper server locations so that the workload on each server is balanced. A naive placement strategy may cause "hot spot" servers that are overloaded, while other servers are under-utilized. Another important issue is choosing the optimal number of replicas. The denser the distribution of replicas is, the shorter the distance a client site needs to travel to access a data copy. However, maintaining multiple copies of data in Grid systems is expensive, and therefore, the number of replicas should be bounded. Clearly, optimizing access cost of data requests and reducing the cost of replication are two conflicting goals. Finding a good balance between them is a challenging task. Finally, we also consider the issue of service locality. Each user may specify the minimum distance he can allow from him to the nearest data server. This serves as a locality assurance that users may specify, and the system must make sure that within the specified range there does exist a server to answer the request.

We propose efficient algorithms for selecting optimal locations for placing the replicas so that the workload among these replica is balanced. Also when given the data usage and service locality requirement from each user site and the maximum workload allowed for each replica server, our algorithm efficiently determines the minimum number of replicas required, as well as their locations.

The rest of the paper is organized as follows. Section 2 describes our data grid model, and formally define our replica placement problem. Section 4 presents our replica placement algorithms, and provides theoretical analysis for them. Section 5 concludes and addresses several open questions and future works.

2 Model

We assume a *hierarchical data grid* model in this paper, due to its resemblance to hierarchical grid management, usually found in current grid systems [9, 6, 11, 18]. For example, in LCG (World-Wide Large Hadron Collider Computing Grid) [9] project 70 institutes from 27 countries form a grid system. The system is organized as a hierarchy, with CERN (the European Organization for Nuclear Research) as the root, or tier-0 site. There are 11 tier-1 sites directly under CERN that help distribute data obtained from Large Hadron Collider (LHC) at CERN. Other tier-2 sites in LCG hierarchy receive data from its corresponding tier-1 site. The entire LCG grid can be represented as a tree structure. In the forthcoming EGEE/LCG-2 grid there will be 160 sites from 36 countries in this tree structure. As a result, this paper focuses on tree topology.

We use a tree T to represent a data grid system. The root of the tree, denoted by r, is the *hub* of the data grid. We assume that the database is located at the hub, and a database replica can be placed in any tree nodes other than the hub. All the leaves of the tree are local sites where user can issue requests to access the databases stored in this data grid system.

A user of a local site at the leaf accesses a database as follows. First he tries to locate a database replica locally. If a replica is not present, he goes to the parent node to find if a replica is there. Namely a user request goes up the tree and uses the first replica encountered along the path towards the hub. We also assume that each request has a *range limit*. This serves as a locality assurance so that the request must be served by a replica, or the hub, within a fixed number of steps towards the root. For example, in Figure 1, the user at node a tries to access a data within a range limit 2. The user could not find the data locally, so he tries the parent node b, where the data is not available either. Finally the request reaches node c, and is served by the replica there. The replica is within the range limit of a so the access is valid. If the range limit of leaf a is 1 instead, we fail to serve the request by this arrangement of replica, since there is no server placed at either a or b. Formally we define that a request can *reach* a server if the number of communication links between it and the nearest replica along the path to the hub is no more than its range limit.

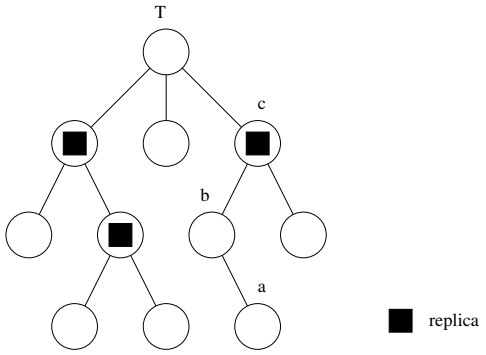

Figure 1. A data grid tree T.

The goal of our replica placement strategy is to place the replicas so that various objectives are satisfied. For example, if we can accurately estimate the usage frequency of user from a leaf node towards a particular data, then we can determine where to place a given number of replicas so that the maximum amount of data a replica server has to serve is minimized. Or, if we fix the total amount of workload a replica server can handle, then we can decide the minimum number of replicas and the best locations to place them.

Next, we formally define the goals of our replica placement strategy. Let T be the data grid system tree, V be the set of nodes of T. Let v be a leaf in V, then $w(v)$ is the amount of workload from v, and $l(v)$ is its range limit. Note that for ease of explanation we place workload only on the leaves. It is trivial to generalize the results of this paper to cases where the internal nodes can also introduce workload.

Let R be a subset of V, and a replica is placed in each node of R. According to the data grid access model mentioned earlier, for each leaf v we define its server to be the first node in R that v encounters when its request goes up towards the root of T. This server node is denoted by $s_R(v)$. The *workload* on a tree node v is defined to be the sum of the data requests for which v is its server, i.e. $w_R(v) = \sum_{s_R(l)=v} w(l)$. However, if the distance between a leaf v and its server $s_R(v)$ is larger than its range limit $l(v)$, the workload of the server is set to infinity.

From the definitions above we can define that a replica set R is *range feasible* if and only if none of the tree node has infinite workload, i.e., every request can reach a server (or the hub) within its range limit. In addition, let the maximum workload induced by R be the maximum workload from all nodes of R *and the hub*. The reason we include the hub is that all the data requests unanswered by R will eventually be answered by the hub. A replica set R is *workload W feasible* if and only if the maximum workload of tree nodes due to R is no more than W. Note that this definition implies that a workload W feasible replica set *is* also range feasible. Now we can formally define our objectives.

- Given the number of replica k, find a range feasible R replica set so that the maximum workload among all nodes of R and the hub is minimized.

- Given the amount of workload a replica or the hub can handle (denoted as W), find the minimum cardinality R that is workload W feasible.

We will refer to the first problem as **MinMaxLoad**, and the second problem as **FindR**.

3 Related Works

This section summarizes the related work in placing replica in tree topology. The first set of models allow the request to go up and down the tree for the nearest replica. For example, Wolfson and Milo [23] suggested a model in which no limit is set for the server capacity. The read cost is the number of hops from a request to its server. The update cost is proportional to the subtree that spans all the replicas. The goal is to minimize read and update cost. Kalpakis et. al. [13] suggested a model in which each server has capacity limit and each site has different building cost. The read cost is defined as the product of the amount of data transfer multiplied and the path length. The goal is to minimize the summation of read, update and site costs. Unger and Cidon [22] suggested a similar model but without server capacity limit. Guha et. al. [10] suggested a model in which there are a known number of servers in the tree, each with equal capacity. There is no read, write, or site building costs, and the goal is to assign the request to a server (not necessarily the nearest one), so that the maximum distance from a client to its assigned server is minimized. Korupolu et. al. [14] suggested a model in which the read cost is slightly different from other models. The data access must go from the client, to the least common ancestor of the client and the replica, the to the replica.

The second set of models only allow the request to search for the replica towards the root of the tree. For example, Jia et. al. suggested a model in which no server capacity nor site building cost is set. The read cost is defined as the product of access path length and the amount of data, and the update cost is defined as the sum of link cost of the subtree from the root to all replicas. The goal is to minimize the sum of read and update cost. Cidon et.al. [5] suggested a similar model in which a replica is associated with a site building cost, but there is no update cost. The goal is to minimize the sum of read cost and storage cost Tang and Xu [20] later described a model similar to our model described in Section 2. There is a range limit on the number of hops a request from its assigned replica. however, there is no server capacity limit in this model. The goal is to find a feasible solution and minimize the sum of update cost and storage cost.

Our model focus on the tree topology in which the requests can only go upwards towards the root. In real-life grid system like LCG [9], the requests go from tier-2 to tier-1, then to tier-0 site in the search of data. In addition, the grid hierarchy usually reflects the the structure of administrative organization, or the geographic locality, so the assumption of having requests

going up towards the root is realistic.

Our model differs from the previous model (except the model in [20]) since each client can assign its own acceptable quality of service, in terms of the number of hops towards to root of the tree. This is an important requirement since different users may require different levels of service quality. We model this as the range limit so that the placement algorithm must make sure that all the replicas are placed in such a way to satisfy all the quality of service requirements. Finally, our model emphasizes on workload balancing so we place a capacity limit on the amount of data a replica can serve. On the other hand, the model in [20] emphasizes update and site building costs.

4 Algorithms

This section describes our algorithms that solve **MinMaxLoad** and **FindR**. We will solve the **FindR** first, and then use that algorithm to solve **MinMaxLoad**.

4.1 FindR

The problem **FindR** can be stated as follows. Given a data grid tree T, the workload and the range limit on its leaves, and a maximum workload W, find a workload W feasible replica set R with minimum cardinality. We use $m(T, W)$ to denote this minimum cardinality. We also define that a replica set R is *optimal* for T and workload W if R is workload W feasible, with only $m(T, W)$ replicas, and minimizes the workload *on the hub*. We will drop the phrase "for workload W" when the context clearly indicates the workload bound. This section shows how to compute an optimal replica set R for a data grid tree T.

4.1.1 Contribution function

We first define several terminologies. Consider a data grid tree T with r as the root. Let v be a node in T, and we use $t(v)$ to denote the subtree rooted at v, and $t'(v) = t(v) - v$, namely the forest of subtrees rooted at v's children. Also we use $a(v, i)$ to denote the i-th ancestor of node v while traversing towards the root of T.

We now define a contribution function C. $C(v, i)$ indicates the minimum amount of workload on the node $a(v, i)$ contributed by $t(v)$, by placing $m(t(v), W)$ replicas in $t'(v)$ and none at $a(v, j)$ for $0 \leq j \leq i$. By definition $C(v, 0)$ is the amount of workload on a node v due to an optimal replica set for $t(v)$. For ease of explanation if there is no replica set for $t(v)$ with cardinality $m(t(v), W)$ that could control the workload on $a(v, i)$ within W, $C(v, i)$ is set to infinity. For example, if v is a leaf, $C(v, i)$ is $w(v)$ when $i \leq l(v)$, and infinity when $i > l(v)$.

Figure 2 illustrates an example of C function. The optimal replica set requires three replicas for T when the workload W is 60, i.e., $m(t(r), 60) = 3$. The replicas should be placed at leaves a, b and c to minimize the workload on r to $C(r, 0) = 35$. However, when we tried to minimize the workload on $s = a(r, 1)$ by placing only three servers, we can only achieve $C(r, 1) = 55$ by placing replicas at nodes a, b and e. We need to place a replica on node e since its range limit is only 1. Now if we consider the workload on $t = a(r, 2)$, since we could not limit its workload within 60 by placing only three replicas in $t'(r)$, $C(r, 2)$ is set to infinity.

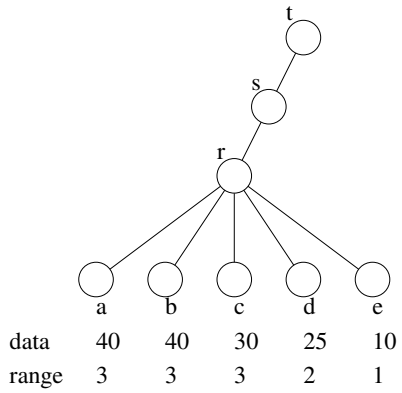

Figure 2. The contribution function.

4.1.2 Bottom-up Computation

Now we describe a bottom-up process for computing C and m functions for every node in a data grid tree. By definition, if v is a leaf, $C(v, i)$ is $w(v)$ when $i \leq l(v)$, and infinity otherwise. Now we would like to compute the C and m function for an internal node r that has children $v_1, ..., v_n$. Since the process is bottom-up, we assume that we know all the C and m functions of $v_1, ...v_n$. The following theorem establishes the relation between the optimal replica sets for a tree and any of its subtrees.

Theorem 1 *Consider a data grid tree T, a node v in T, and a workload W. There exists an optimal replica set R for T with workload limit W so that $|R \cap t'(v)| = m(t(v), W)$*

Proof. By definition $t(v)$ requires at least $m(t(r), W)$ replicas to be placed in $t'(r)$ to make sure that the workload on v is within W. As a result there does not

exist an optimal replica set R for T that could place less than $m(t(v), W)$ replicas in $t'(v)$.

Now if an optimal replica set for T places more than $m(t(v), W)$ replicas in $t'(v)$, it places at least $m(t(v), W) + 1$ replicas. In that case while we are constructing an optimal replica set for T, we simply place $m(t(v), W)$ replicas according to an optimal replica set for $t(v)$, and place one extra replica at v. The resulting new replica set for T will not introduce any extra workload, and therefore is also an optimal solution for T. ■

Theorem 1 suggests that for a node v with children $v_1, ..., v_n$, there exists an optimal replica set R with workload limit W so that $|R \cap t'(v_j)| = m(t(v_j), W)$, for $1 \leq j \leq n$. As a result in order to find the optimal replica set for $t(v)$ we need to place the least number of replicas at v_j's so that the sum of their $C(v_j, 1)$ is at least $\sum_{1 \leq j \leq n} C(v_j, 1) - W$. This can be easily done by repeatedly placing a replica at the remaining v_j that has the largest $C(v_j, 1)$, until the workload of v is within W. Let this set of v_j's be $e(v, 0)$, which is the set of children of v that each of them has to be placed a replica to minimize the workload on $a(v, 0) = v$. We have $m(t(v), W) = \sum_{1 \leq j \leq n} m(t(v_j), W) + |e(v, 0)|$, and $C(v, 0) = \sum_{v_j \notin e(v,0)} C(v_j, 1)$.

After knowing $m(v, W)$, we want to compute $C(v, i)$ for $i > 0$.

Theorem 2 *Consider a data grid tree T, a node v in T with children $v_1, ..., v_n$, and a workload W. There exists a replica set R so that $|R| = m(T, W)$, R minimize the total workload due to R from $t'(v)$ on $a(v, i)$ for $i \geq 1$, and $|R \cap t'(v_j)| = m(t(v_j), W)$.*

Proof. The proof is similar to Theorem 1. Consider Figure 3. First $|R \cap t'(v_j)|$ could not possibly be less than $m(t(v_j), W)$, otherwise the workload on v_j will already be greater than W. If R has more than $m(t(v_j), W)$ replicas in $t'(v_j)$, we can place only $m(t(v_j), W)$ according to any optimal replica set for $t(r_i)$, and place one replica at v_j. The resulting replica set will not contribute more workload on $a(v, i)$. ■

An important implication of Theorem 2 is that when we compute $C(v, i)$ for $i > 0$, we can be certain that there exists a set of replicas that can minimize the total workload on $a(v, i)$, with a *known* number of replicas in each of $t'(v_j)$ (i.e., $m(t(v_j), W)$). By definition $C(v, i)$ is the the minimum amount of workload on the ancestor $a(v, i)$ contributed by $t(v)$ by placing $m(t(v), W)$ replicas in $t'(v)$, and none at $a(v, j)$ for $0 \leq j < i$ (see Figure 3 for an illustration). Since we know the number of replicas in each of $t'(v_j)$, we know there exists an R that

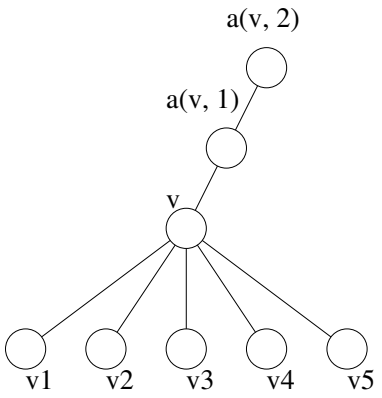

Figure 3. The computation of contribution function $B(v, i)$ for $i > 0$.

can minimize the total workload $a(v, i)$ by placing extra $m(t(r), W) - \sum_{1 \leq j \leq n} m(v_j, W)$ replicas among r_i. Now it is clear that by choosing $m(t(r), W) - \sum_{1 \leq j \leq n} m(v_j)$ v_j's that have the largest $C(v, i+1)$ (denoted as $e(v, i)$), we can derive $C(v, i) = \sum_{v_j \notin e(v,i)} C(v_j, i + 1)$.

4.1.3 Top-down replica placement

We now place replicas from top to the bottom of tree recursively. Consider a node v, which has n children $v_1, ..., v_n$. Our goal is to place replicas so as to minimize the workload on $a(v, i)$ by placing $m(v, W)$ replicas in $t'(v)$, therefore our recursion starts from the root and with $i = 0$. From the discussion in Section 4.1.2, we know we can accomplish this by placing replicas in $e(v, i)$ – the subset of $\{v_1, ...v_n\}$ that each of them has to be placed a replica in order to minimize the workload on $a(v, i)$.

We consider two cases. In the first case we consider the set of v_j's that are in $e(v, i)$. Let A denote the set of children of these v_j's. We can start the recursion from each node of A with i set to 0, since we know that every node in $e(v, i)$ now has a replica. In the second case we consider the set of v_j's that are *not* in $e(v, i)$. From theorem 2, we know that if we want to minimize their contribution to $a(v, i)$, we just need to focus on these replica sets that have $m(v_j, W)$ replicas in $t'(v_j)$. In addition, we know that we can also assume that there will be *no* replica in any of them. As a result we simply retrieve $e(v_j, i + 1)$ – the subset of v_j's children that should be placed an extra replica to minimize the workload contribution on $a(v,_i)$.

The pseudo code of this recursive top-down replica placement procedure is in Figure 4. The parameter i indicates the level of recursion towards the node whose workload we want to minimize.

```
Place-replica(v, i)
{
   if i is a leaf
      return;

   place a replica at each node of e(v, i);
   for each child c of v {
      if c is in e(v, i)
         Place-replica(c, 0);
      else
         Place-replica(c, i + 1);
   }
}
```

Figure 4. The pseudo code of the recursive top-down replica placement.

Consider the example in Figure 5. The recursion starts at a with $i = 0$. Suppose we know that we need to place a replica at b and c. Now we only need to recursively perform the top-down replica placement at the children of b and c, namely f, g, h, and i. After placing the replicas at nodes b and c, we know that there will be no replica placed at d and e. Now consider the three subtrees of d. We know there exists a replica set that can minimize the contribution to a, with $m(j, W)$ replicas in $t'(j)$, $m(k, W)$ replicas in $t'(k)$, and $m(l, W)$ replicas in $t'(l)$. We also know that this replica set has *no* replica in d. As a result we simply retrieve $e(d, 1)$ – the subset of $\{j, k, l\}$ that should be placed an extra replica to minimize the workload contribution on a.

We will refer to the entire replica-placement algorithm as *PlaceReplica*.

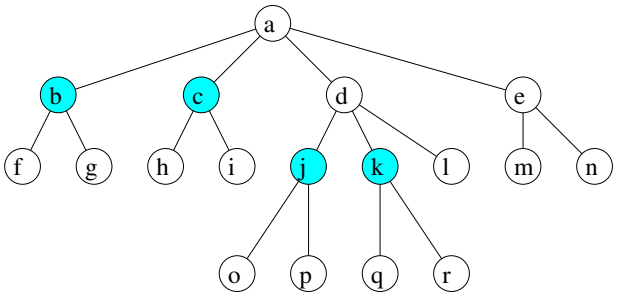

Figure 5. An example of top down replica placement. A shaded tree node indicates a replica.

4.1.4 Time complexity analysis

We analyze the time complexity of *PlaceReplica* by focusing on the bottom-up computation of C contribution function, since it dominates the total computation time. For each tree node v we need to compute its $C(v, i)$, up to $i = L$, where L is the maximum range limit among all nodes. Let the number of children of v be n, then the computation requires $n \log n$, if we sort the C functions from all of v's children. Note that this requires L sorting since a child v_j with a larger $C(v_j, i)$ function value than another child v_k does not mean v_j will have a larger C values than v_k for other values of i. As a result the computation cost of C for v is $Ln \log n$. The total cost of computing C functions for all nodes is therefore $LN \log N$, where N is the number of nodes in the tree.

4.2 MinMaxLoad

We now derive an algorithm *BinSearch* for the **MinMaxLoad** problem. The algorithm *BinSearch* finds the replica set by "guessing" the maximum workload W with a binary search. We guess a value of W as the maximum workload on the replicas and the hub. If the algorithm *PlaceReplica* could not find an optimal replica set within cardinality k, we increase the value of W; otherwise we reduce it. We estimate an upper bound U on the workload for every leaf, therefore the total amount of workload is bounded by $O(NU)$. It is easy to see that after $O(\log N + \log U)$ calls of *BinSearch*, we will be able to find the smallest value of W by which only k replicas suffices. The total execution time of *BinSearch* is therefore $O(\log N + \log U)(LN \log N)$. Because the bound on the workload, U, is usually represented by a 32 bit integer, the total execution time can be bounded by $O(LN \log^2 N)$.

Theorem 3 *The algorithm* BinSearch *finds the optimal replica set for* **MinMaxLoad** *in time* $O(\log N + \log U)(LN \log N)$*, where* N *is the number of tree nodes in the data grid,* L *is the maximum range limit. and* U *is the maximum workload. When* U *is a bounded constant, the time complexity is* $O(LN \log^2 N)$.

5 Conclusion

This paper addresses the issues of placing database replicas in Data Grid systems with locality assurance. Each request specifies a workload it requires, and a distance within which a replica must be found. We propose efficient algorithms that select strategic locations for placing the replica so that the workload among

these replicas is balanced, and the service locality required by each data request is guaranteed.

We formulate two problems: **MinMaxLoad** and **FindR**, and derive efficient algorithmic solutions for them. Based on the estimation of data usage and the given locality requirement from various sites, our algorithm efficiently determines the locations of replica if both the number of replicas and the maximum allowed workload for each replica have been determined. Another algorithm can determine the number of replicas needed to ensure that the maximum amount of workload on every replica is below a certain threshold.

One open question for the replica placement problem is how to determine the replica location when the network is a general graph, instead of a tree. It is possible that we may need to consider other graphs, (e.g. planar graphs), and derive efficient algorithms for them. In addition, in the current hierarchical Data Grid model, all the traffic may reach the root, if it is not answered by a replica. This additional constraints introduce additional complexity for the design of efficient algorithm for replica placement in grid systems when network congestion is one of the objective function to be optimized.

Acknowledgment The authors would like to acknowledge the National Center for High-Performance Computing in providing resources under the national project "Taiwan Knowledge Innovation National Grid".

References

[1] J. H. Abawajy. Placement of file replicas in data grid environments. In *ICCS 2004, Lecture Notes in Computer Science 3038*, pages 66–73, 2004.

[2] M. M. Bae and B. Bose. Resource placement in torus-based networks. *IEEE Transactions on Computers*, 46(10):1083–1092, October 1997.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, (23):187–200, October 2000.

[4] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe. Wide area data replication for scientific collaborations. In *In Proceedings of the 6th International Workshop on Grid Computing*, November 2005.

[5] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40(2):205–218, 2002.

[6] W. B. David. Evaluation of an economy-based file replication strategy for a data grid. In *International Workshop on Agent based Cluster and Grid Computing*, pages 120–126, 2003.

[7] W. B. David, D. G. Cameron, L. Capozza, A. P. Millar, K. Stocklinger, and F Zini. Simulation of dynamic grid rdeplication strategies in optorsim. In *In Proceedings of 3rd Intl IEEE Workshop on Grid Computing*, pages 46–57, 2002.

[8] M.M. Deris, Abawajy J.H., and H.M. Suzuri. An efficient replicated data access approach for large-scale distributed systems. In *IEEE International Symposium on Cluster Computing and the Grid*, April 2004.

[9] Worldwide LHC Computing Grid. http://lcg.web.cern.ch/lcg/.

[10] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *J. Algorithms*, 48(1):257–270, 2003.

[11] W. Hoschek, F. J. Janez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. In *In Proceedings of GRID Workshop*, pages 77–90, 2000.

[12] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, June 2001.

[13] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):628–637, 2001.

[14] M. Korupolu, C. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. *Journal of Algorithms*, 38(1):260–302, 2001.

[15] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman. Data replication strategies in grid environments. In *In Proceedings of 5th International Conference on Algorithms and Architecture for Parallel Processing*, pages 378–383, 2002.

[16] R. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan. *I. Foster and C. Kesselman edited,*

The Grid: Blueprint for a Future Computing Infrastructure, chapter Data intensive computing. Morgan Kaufmann PUblishers, 1999.

[17] K. Ranganathan, A. Iamnitchi, and I.T. Foste. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *In 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 376–381, 2002.

[18] K. Ranganathana and I. Foster. Identifying dynamic replication strategies for a high performance data grid. In *In Proceedings of the International Grid Computing Workshop*, pages 75–86, 2001.

[19] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in data grids. In *In 10th IEEE Symposium on High Performance and Distributed Computing*, pages 305–314, 2001.

[20] X. Tang and J. Xu. Qos-aware replica placement for content distribution. *IEEE Transactions on Parallel and Distributed Systems*, 16(10), October 2005.

[21] N.-F. Tzeng and G.-L. Feng. Resource allocation in cube network systems based on the covering radius. *IEEE Transactions on Parallel and Distributed Systems*, 7(4):328–342, April 1996.

[22] O. Unger and I. Cidon. Optimal content location in multicast based overlay networks with content updates. *World Wide Web*, 7(3):315–336, 2004.

[23] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Trans. Database Syst.*, 16(1):181–205, 1991.