A List-based Strategy for Optimal Replica Placement in Data Grid Systems

Yi-Fang Lin*†, Jan-Jan Wu†, and Pangfeng Liu*‡
* Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
Email: {d92009, pangfeng}@csie.ntu.edu.tw
† Institute of Information Science, Academia Sinica, Taipei, Taiwan
Email: {ice, wuj}@iis.sinica.edu.tw
‡ Graduated Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan

***Abstract*—Data replications is a typical strategy for improving access performance and data availability in Data Grid systems. Current works on data replication in Grid systems focus on the infrastructure for data replication and the mechanism of replicas creation and deletion. The important problem of choosing suitable locations for placing replicas in Data Grids has not been fully studied.**

This paper addresses replica placement problem in Data Grids when given a sequence of *priority lists* that specify the forwarding policies for data requests. We propose the concept of priority list to address two issues. First, a user may have limited authority in accessing the resources, and thus his/her data requests should be prohibited from accessing some of the sites. Second, a static policy may not satisfy a data request with special requirements (e.g. quality of service requirement). In this priority-list-based model we propose a placement algorithm that finds optimal locations for replicas so that the workload among the replicas is balanced. We also propose an algorithm that determines the minimum number of replicas when the maximum workload capacity of each replica is given.

a) Keywords: Data Grids, replica placement, priority list, dynamic programming.

I. Introduction

Data Grids provide geographically distributed storage resources for complex computational problems that require the evaluation and management of large amounts of data [3], [12], [17]. With the high latency of the wide-area networks that underlie most Grid systems, and the need to access/manage several petabytes of data in Grid environments, data availability and access optimization have become key challenges that must be addressed.

An important technique that speeds up data access in Data Grid systems replicates the data in multiple locations so that a user can access it from a site in his vicinity. It has been shown that data replication not only reduces access costs, but also increases data availability in many applications [12], [18], [15]. Although a substantial amount of work has been done on data replication in Grid environments, most of it has focused on infrastructures for replication and mechanisms for creating/deleting replicas [4], [7], [6], [8], [12], [15], [19], [18], [20]. We believe that, to obtain the maximum benefit from replication, strategic placement of the replicas is also necessary.

A number of early works addressed placement of data replicas in parallel and distributed systems with regular network topologies such as hypercubes, torus, rings, and trees [2], [13], [22]. These networks posses many attractive mathematical properties that enable the design of simple and robust placement algorithms, but they cannot be directly applied to Data Grid systems due to hierarchical network 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 called `Proportional Share Replication` for the placement problem, but it does not guarantee an 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 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 to choose 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 a large number of data copies 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, so finding a good balance between them is a challenging and interesting task.

Most of the current work assume a static forwarding policy for all data requests [1], [2], [13], [22]; that is, the forwarding destination of each site is fixed regardless the characteristics of the data requests. Such static policy may not be applicable to many grid systems because of the following reasons. First, a user may have limited authority in accessing the resources, and thus his/her data requests may be prohibited from accessing some of the sites. Second, a static policy may not satisfy a data request with special requirements (e.g. quality of service requirement). Finally, the sites a request has visited may influence the subsequent forwarding path of the request. Due

to these reasons, we believe that a suitable forwarding policy should allow individual requests to have varied forwarding paths. To accomplish this goal, we facilitate a data request with a sequence of *priority lists* to allow more flexible request forwarding policies. In other words, the Grid administrators can provide the *priority lists* of data requests for the variant policies.

In this paper, we focus on a tree-like-structured grid model, which reflects the hierarchical structure in many grid systems [9], [6], [12], [19]. This hierarchy may come from the network topology or administration institution. The hierarchical Data Grid model is also one of the most important common architecture in current use [1], [6], [12], [19], [10]. For example, in the LCG (WorldWide Large Hadron Collider Computing Grid) [9] project, 70 institutes from 27 countries form a grid system 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 the Large Hadron Collider (LHC) at CERN. Meanwhile, each tier-2 site in the LCG hierarchy receives data from its corresponding tier-1 site. In the future, LCG is expected to extend to more levels of tiers. The entire LCG grid can be represented as a tree structure. In the forthcoming EGEE/LCG-2 Grid, the tree structure will comprise 160 sites in 36 countries.

We study replica placement in the tree-like model augmented with *priority lists*-based forwarding policy. Under this model, we propose an efficient algorithm that finds the optimal locations for placing the replicas so that the workload among these replica is balanced. We also propose an algorithm that determines the minimum number of replicas when the maximum workload capacity of each replica server is given.

The rest of the paper is organized as follows. Section II describes our data grid model, and formally defines our replica placement problem. Section III summarizes related work in replica placement in trees. Section IV presents our replica placement algorithms and their theoretical analysis. Section V gives concluding remarks and directions for future research.

II. Model

We propose a tree-structured data grid model augmented with *priority lists* to exploit pragmatic request forwarding policies in many Data Grid systems. We first formally define the notion of priority list and replica placement, and then describe the replica placement problems in this model.

A. The Priority List Model

A Data Grid consists of a node set $V = \{n_1, n_2, \ldots, n_n\}$ that represents n data grid nodes. Each node in the Data Grid can request data, and each request uses a *priority list* to indicate the sites from which this request will ask for the data, in the order specified in the priority list. We will use the term "request" and "priority list" interchangeably in the rest of the paper. We assume that there are m data requests in total, and they form a *priority list sequence* denoted by $L = (l_1, \ldots, l_m)$. The i-th priority list in L is $l_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,k_i})$, where k_i is the number of nodes in l_i, and each $p_{i,j}$, for $1 \le i \le m, 1 \le j \le k_i$, is a node in V.

A *server set* is a subset of V that have data replicas. A server set S *satisfies* a request l_i if and only if the intersection of S and l_i is not empty. To be more specific, the request l_i will receive the data from the first server in its priority list that also appears in S. Formally, a node c is the *server* of a request l_i under a server set S, denoted as $c = ser(S, l_i)$, if c is the first server appearing in l_i that also appears in S.

$$ser(S, l_i) = \begin{cases} \emptyset & p_{i,j} \notin S, \quad 1 \le j \le k_i \\ p_{i,j^*} & j^* = \min_j \{j | p_{i,j} \in S\} \end{cases} \tag{1}$$

For example, we consider a data grid system with a node set $V = \{n_1, n_2, \ldots, n_{10}\}$, three requests $l_1 = (n_1, n_5, n_3, n_6)$, $l_2 = (n_9, n_2, n_7)$ and $l_3 = (n_5, n_{10})$, and a server set $S = \{n_2, n_3, n_5\}$. The servers of requests l_2 and l_3 under S are n_2 and n_5, respectively, since each of the two requests has only one server in S. The server of l_1 is n_5 since it is the first server in l_1 that is also in S, according to Equation 1.

B. Replica Placement Problem

Given a server s from a server set S and a priority lists sequence L, the *workload* of s is the number of requests in L that s satisfies. This workload is denoted by $load(L, S, s)$ as in Equation 2. The workload of a server set S is defined as the *maximum* workload among its servers, as denoted by $Load(L, S)$ in Equation 3.

$$load(L, S, s) = |\{l | l \in L, s = ser(S, l)\}| \tag{2}$$

$$Load(L, S) = max_{s \in S} load(L, S, s) \tag{3}$$

A server set S is *feasible* if S satisfies all m requests in L. If a server set S is *feasible* and the workload $Load(L, S)$ is no more than W, then S is *workload-W-feasible*.

For example, consider the same V and the same priority list sequence L in the previous example, and two server sets $S_1 = \{n_2, n_3, n_5\}$ and $S_2 = \{n_1, n_2\}$. According to the definition of server set feasibility, S_1 is feasible but S_2 is not, since S_2 cannot satisfy l_3. The workloads of n_2, n_3 and n_5 are 1, 0 and 2, respectively, when given the server set S_1.

The goal of our replica placement strategy is to place the replicas so as to satisfy the following two objectives. First, given a priority lists sequence L, how to determine where to place a given number of replicas so that the maximum server workload is minimized? Second, if we fix the maximum workload of each server, how to determine the minimum number of replicas and the best locations to place them? We formulate these two replica placement problems as follows.

- The **LoadBalance** problem is, when given the number of replica k, find a feasible server set S of k servers so that the maximum workload among all servers in S is minimized.
- The **PlaceR** problem is, when given the maximum workload of each server (denoted by W), find the minimum cardinality server set S that is workload-W-feasible.

C. The Sibling Tree Model

The rest of the paper will focus on a special priority list model – the *sibling tree* model. We focus on the sibling tree model because it resembles the hierarchical grid management found in many Grid systems [9], [6], [12], [19].

Abawajy reported a hierarchy communication model in [1]. In this hierarchy model, the nodes are connected as a tree, which corresponds to hierarchical grid management found in many Data Grids systems. In this hierarchy model, the requests issued on a node v will be forwarded towards the root of the tree via the unique path between v and the root.

The sibling tree model is similar to the hierarchy communication model in [1], with additional communication channels between siblings. That is, in addition to the communication channel from each tree node to its parent, we also have communication channels between siblings. The reason for these additional links is that, in a hierarchical grid, a request would usually visit all of the nodes managed by the same parent before it is forwarded to the parent. Note that the communication links between siblings can be logical – one node can always reach its sibling via the parent by at most two hops. Nevertheless, in the rest of the paper, we will assume that the links between siblings are physical.

The edges in a sibling tree consist of *tree edges* and *sibling edges*, therefore a sibling tree is the union of a tree edge set T and the ring edge set E among siblings. We also assume that all edges in the sibling ring are *one-way* so that messages can only travel along the ring in one direction.

For each node in a sibling tree, we define a *sibling path*. The sibling path of a node v, denoted by $SP(T, v)$, is the path the requests from v will travel while searching for the requested data. If a parent has q children $c_1, \ldots, c_q$ where c_i is the i-th child, the sibling path of c_i is the concatenation of the path $(c_i, c_{i+1}, \ldots, c_q, c_1, c_2, \ldots, c_{i-1})$, and the sibling path of the parent p, i.e., $SP(T, p)$. The request will travel through all children of p (via the one-way sibling edges mentioned earlier), then proceed to the sibling path of the parent p.

This paper focuses on the replica placement problem where each request (priority list) from a node v is a *prefix* of the sibling path of v. In other words, all requests from v will first travel along the sibling ring looking for the data. If none of its siblings has the data, the request searches along the sibling path of its parent.

The sibling-path routing behavior mentioned earlier enforces a *quality of service* requirement. If a request has k nodes in its priority list, it must be able to find a replica within the first k nodes along its sibling path. Otherwise, this request is *not* satisfied and the server set is *not* feasible. Therefore, the number of nodes in a priority list can be viewed as a QoS requirement for the request. We must find a server set that satisfies all requests, which is equivalent to satisfying the QoS requirement of all requests.

Note that under a feasible server set S in the sibling tree model where every request is satisfied, the requests from the same node will be served by the *same* server. This is because that requests from the same node travel along the prefixes of the *same* sibling path. Consequently, after we are given the a server set S and a priority list sequence L. we can define two functions for each node v – $w(v)$ is the number of the priority lists (requests) that start from v, and $r(v)$ is the shortest length of the priority lists from v. $r(v)$ represents a quality-of-service requirement, since every feasible server set must place a replica no further than the $r(v)$-th node from v along the sibling path of v.

Another implication of this prefix-based routing is that the requests sent from the root must be serviced by a replica at the root. Therefore, we can assume that the priority list of a request from the root contains only the root, and there is an "implicit" replica at the root.

III. Related Works

This section summarizes related work in replica placement in trees. The first set of models allow the request to go up and down the tree for the nearest replica. For example, Wolfson and Milo [24] 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 size of the subtree that spans all replicas. The goal is to minimize the sum of read and update cost. Kalpakis et. al. [13] suggested a model in which each server has capacity limit and each site has different site building costs. The read cost is defined as the product of the amount of data transfer and the path length. The goal is to minimize the sum of read, update and site building cost. Unger and Cidon [23] suggested a similar model but without server capacity limit. Guha et. al. [11] 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, then 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 or site building cost is set. The read cost is defined as the product of access path length and the amount of data. The update cost is defined as the sum of link cost of the size of the subtree from 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 [21] later described a model similar to our quality of service model, but without server capacity limit and the sibling path. There is a range limit on the number of hops between a request and its assigned replica. The goal is to find a feasible solution and minimize the sum of update and storage costs.

Our model focus on a tree-like topology in which the requests 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 structure of administrative organization, or the geographic locality, so the assumption of having requests going up towards the root is reasonable. Moreover, our model allows a request to be forwarded among the siblings before this request goes up towards the root. The reason is that, in a hierarchical grid, a request would usually visit all of the nodes managed by the same parent site before it is forwarded to the parent.

Our model differs from the above mentioned models because it considers all of the following three factors. First, we allow each client to specify its own quality of service requirement, in terms of the number of hops towards to root of the tree. This extension allows different users to specify different levels of service quality. Second, our model addresses the important issue of workload balancing by setting a capacity limit on the amount of data that each replica server can handle. Finally, our model allows sibling links to reflect the reality that a site may request its peer sites before requesting its parent.

IV. The Algorithms

A. PlaceR

The **PlaceR** problem in the sibling tree model can be stated as follows. Given a sibling tree T, a priority list sequence L, and a maximum workload W, find a workload-W-feasible server set S with the *minimum* number of replicas. However, as mentioned in Section II-C, the requests from the root must be answered by a replica at the root, so we will assume that there is an "implicit" replica at the root, and the *minimum* number of replicas in the definition of workload-W-feasible does *not* include this root replica.

To take into consideration this "implicit" replica at the root, we would like to refine the definition of workload-W-feasible as follows. Let T_v be the subtree rooted at v, and $T'_v = T_v - \{v\}$ is the forest of subtrees rooted at the children of v. S is a subset of T'_v that partitions the requests from T_v into two groups. The requests in the first group will reach v and request in the second group are serviced by one of the servers in S. By reaching v we mean that a request will not be able to find any server in S along its sibling path before v. Since we do not wish to have any overloaded replica, including the implicit one at the root, we define that a server set S is workload-W-feasible if the number of requests reaching v or any servers in S is no more than W. In other words, the "implicit" replica at v should not have workload from T_v more than W either.

Similarly we can refine the definition of "optimal servers set" for T_v. A server set S is *optimal* for T_v, L, and W if the followings are true. First, S must be a workload-W-feasible server set with the *minimum* number of servers. Second, S must minimize the number of the requests from T_v that reach v. We use $m(T_v, L, W)$ to denote this minimum number of servers in an optimal server set for T_v. For ease of notation we will drop the phrase "for W and L" when the context clearly indicates the workload bound W and the priority list sequence L. Consequently we will use $m(T_v)$ to indicate the minimum number of servers in an optimal server set for T_v.

1) Contribution function: Let T be a sibling tree rooted at r. For every node v in T, We now define a *contribution* function $C(v, i)$.

Definition 1: Consider a node v and the i-th node z on the sibling path of v. If we put an "implicit" replica at z, then $C(v, i)$ is the *minimum* possible workload on z contributed by T_v under the following three constraints. Note that if there does not exist a workload-W-feasible server set to satisfy the constrains, the contribution function $C(v, i)$ will be set to infinity.

1) There are $m(T_v)$ replicas in T'_v.
2) There is no replica in the first $i - 1$ nodes along the sibling path of v.
3) Each request from T_v must be satisfied by either z or one of the $m(T_v)$ servers in T'_v.

We consider some special cases of the contribution functions. By definition $C(v, 0)$ is the minimum workload of node v due to an optimal server set for T_v. If v is a leaf, the contribution function $C(v, i)$ is $w(v)$ when $i < r(v)$, and infinity when $i \geq r(v)$, where we recall that $w(v)$ is the number of requests from v and $r(v)$ is the maximum number of hops that the requests from v are allowed to search for replica.

We now further generalize the concept of "optimal server set" according to the contribution function. The contribution function defines the workload contribution of a subtree rooted at v towards the i-th node along the sibling path of v. Therefore we define a server set S to be *i-optimal* for T_v if S is a workload-W-feasible set for T_v with minimum number of servers and S *minimizes* the contribution function $C(v, i)$. Let U be the set of the requests from T_v that cannot be satisfied by servers in S. An i-optimal server set S ensures that requests in U reach the i-th node of the sibling path of v, minimize their contributions to the i-th node in the sibling path of v, and S has at most $m(T_v)$ replicas. We use $S(v, i)$ to denote this i-optimal server set. Please refer to Figure 1 for an illustration.

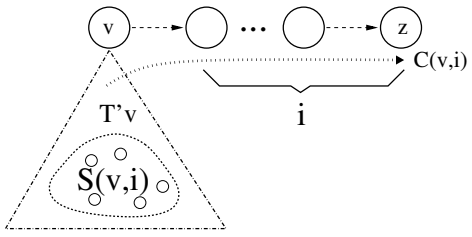

Fig. 1. An illustration of i-optimal server set $S(v, i)$

2) Bottom-up Computation: We now describe a bottom-up process that computes the contribution C and the minimum server cardinality functions m. By definition, if v is a leaf, $C(v, i)$ is $w(v)$ when i is less than $r(v)$, and infinity otherwise. Now consider an internal node v that has children $v_1, \ldots, v_n$. Since this is a bottom-up process we may assume that we have C, m, and the optimal server sets $S(v_j, i)$ for all i and j.

Given the C and m functions of children of a node v, there are two cases we need to consider. The first case is that there is no replica among the children of v, and the second case is that there is at least one replica among them. In the first case we use a greedy method to determine how to combine local optimal solutions of the children into a global optimum. In the second case we transform our problem instance into many problem instances in [16], apply a dynamic programming also described in [16], and find the best solution among these constructed instances. After we compute the m functions for both cases, we then choose the smaller one as the m function for the subtree. Finally we can use the m function to compute the contribution function for the root of the subtree.

3) No replica among the children of v: In the first case we assume that the optimal server set of v does *not* have replica at the children of v. We first determine $m(T_v)$, the minimum number of replica required. Consider any 0-optimal server set $S(v,0)$ for T_v. $S(v,0)$ must have at least $m(T_{v_j})$ in T'_{v_j} for all j, otherwise v_j will be overloaded by requests. Therefore we conclude that $S(v,0)$ must have exactly $m(T_{v_j})$ or $m(T_{v_j})+1$ servers in T'_{v_j}, since adding just one replica at a child of v_j is enough to minimize the workload of v_j due to requests from T'_{v_j}.

If the sum of contribution of T_{v_j} towards v is less than or equal to W, we conclude that any $S(v,0)$ server set could not do better since by definition the contribution function is minimized by the server set $S(v_j,n)$ respectively. Note that we use n (the number of children of v) as the second parameter since node v is the n-th node in the sibling path of any v_j. Consequently, $m(T_v)$ is the sum of all $m(T_{v_j})$.

If the sum of contribution of T_{v_j} towards v is greater than W, we consider two cases. If the server set $S(v,0)$ has $m(T_{v_j})$ replica in T'_{v_j}, then it will not contribute less workload than the server set $S(v_j,n)$, which by definition will contribute the least workload toward v with $m(T_{v_j})$ replicas. Again note that we use n as the second parameter since the node v is the n-th node in the sibling path of any v_j. If the server set $S(v,0)$ has $m(T_{v_j})+1$ replica in T'_{v_j}, then we could replace them with $S(v_j,n)$, which has $m(T_{v_j})$ replicas, plus one at an arbitrary child of v_j. The latter arrangement will contribute zero workload towards v, which could not be outperformed. Consequently, we conclude that there exists an $S(v,0)$ that consists of all $S(v_j,n)$, plus some grandchildren of v.

Now we need to determine which grandchildren of v need to have replicas in order to find a 0-optimal server set for T_v. These extra replicas will be placed into some subtrees of v, and each of these subtrees will place exactly one replica at the grandchild of v. The sum of this extra number of replica, and the sum of all $m(T_{v_j})$, will be equal to $m(T_v)$ in this case.

We will use the following greedy method to place extra replicas. We will choose the subtree that contributes the *most* towards v, place a replica there to reduce the workload that reaches v, and repeat this process until the workload on v does not exceed W. Let Q be the set of roots of the subtrees that we decide to place extra replicas. The final contribution function $C(v,0)$ can be described as Equation 4. Note that for those T'_{v_j} that we decide to place extra replicas, the contribution of T_{v_j} towards v is limited to $w(v_j)$ since the extra replica will intercept all requests from T'_{v_j}, and only the requests of v_j could reach v.

$$C(T_v,0) = w(v) + \sum_{v_j \in Q} w(v_j) + \sum_{u \notin Q} C(u,n). \qquad (4)$$

This greedy method does minimize the workload on v by a server set S^*. We argue that S^* is indeed optimal since if another 0-optimal server set $S(v,0)$ disagrees with S^*, we can replace the subtrees of $S(v,0)$ that are missing from S^* with those that only appear in S^*, and the resulting server set will not increase the contribution function on v.

Figure 2 gives an example in which we should not place replica at any of the children of v. The reason is that if we do so, we need to place additional replicas to avoid the replica we placed at the child of v being overloaded. Instead, we can place only one replica at a grandchild g of v to prevent its requests from reaching v. This will keep the load on v below W, since we have prevented some requests, now served by g, from reaching v.

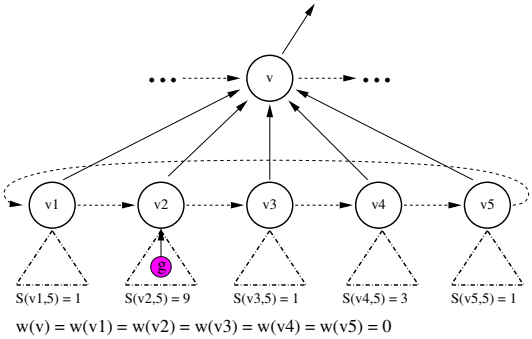

Fig. 2. An example showing that a single replica at a grandchild of v is sufficient to keep v from being overloaded. The capacity limit is set to 10.

Now we have determined $m(T_v)$ and $S(v,0)$, but we still need to determine all $S(v,i)$'s. This process is similar to the one described above. We argue that any $S(v,i)$ will have at least $m(T_{v_j})$ replicas in each of T'_{v_j}, therefore we can "standardize" $S(v,i)$ into a form of the union of all $S(v_j,i+n)$, plus additional replicas at some grandchildren of v. We apply the similar greedy method to find these grandchildren in order to minimize the contribution towards the i-th node on the sibling path of v. Let Q be the set of those subtrees that have extra replicas, then we have Equation 5.

$$C(T_v,i) = w(v) + \sum_{v_j \in Q} w(v_j) + \sum_{u \notin Q} C(u,i+n) \qquad (5)$$

4) Replicas exist among the children of v: We now consider the case that there is at least one child of v that has replica. Please refer to Figure 3 for an example. In this example node v has five children v_1, v_2, v_3, v_4, and v_5. The maximum workload W is 10. The number of requests from each child

is 1. The quality-of-service parameters of five children $r(v_1)$, $r(v_2)$, $r(v_3)$, $r(v_4)$ and $r(v_5)$ are $7, 2, 2, 7$ and 3 respectively. Consequently $C(v_1, k)$ is 7 for $0 < k < 7$, $C(v_2, k)$ is 6 for $0 < k < 6$, $C(v_3, k)$ is 2 for $0 < k < 2$, $C(v_4, k)$ is 9 for $0 < k < 7$, and $C(v_5, k)$ is 4 for $0 < k < 3$. Since the quality-of-service parameters of some children of v (e.g. v_2, v_3 and v_5) are smaller than the number of the children of v, we must place at least one replica among the children of v.

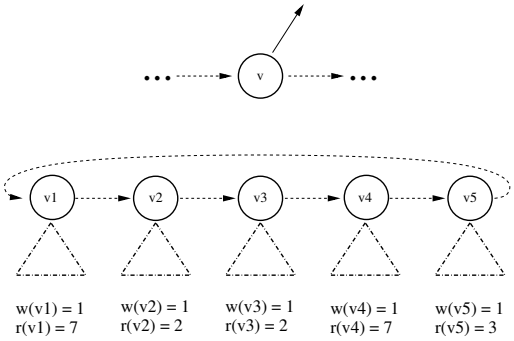

Fig. 3. An example where at least one child of v must have a replica. The server capacity is 10.

We will transform this case into a problem instance described in [16], and use an algorithm described in [16] to solve this case. The communication model in [16] can be described as follows. We are given a tree in which every node can issue requests. There are several servers in the tree. Each request has a workload and and a *range limit*. The requests will go toward the root of the tree, and is served by the first server it encounters. In addition, each request must reach a server within the number of hops specified by the range limit. We will use *range limit model* to denote this setting [16]. The question is, how do we place the minimum number of servers to serve all requests.

Liu et. al. [16] use a dynamic approach to solve the replica placement problem for this range limit model. A contribution function is defined to be the minimum possible workload contribution by a subtree rooted at v towards an ancestor of v. The contribution function is computed in a bottom-up phase using all the contribution functions of the children of v.

In order to compute the contribution function in our model, we consider n cases, where the k-th case is to have v_k as the root of a subtree structure illustrated in Figure 4. This figure illustrates both the tree edges that form the subtrees from T_{v_1} to T_{v_n}, and the ring edges that are between the children of v. We assume in this case that v_k has a replica so every request will eventually stop at v_k if not answered by other servers, and no request can reach v. Consequently the best server set can be found by examining all these n cases, and choose the optimal one.

We now consider the case where v_k is the root of the tree structure. For each subtree T_{v_j} we add a dummy node d_j as the root of T'_{v_j}, and set the parent of d_j to be v_j. Please refer to Figure 4 for an illustration.

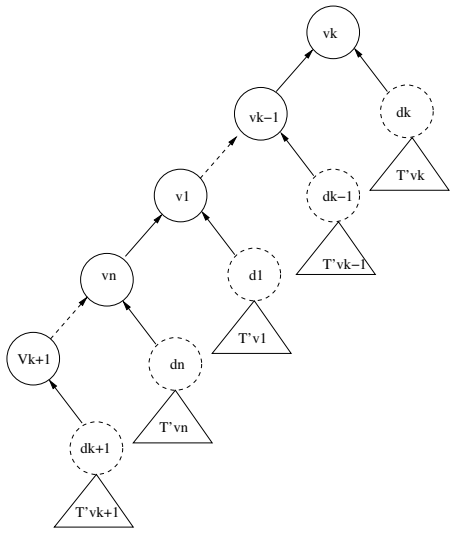

Fig. 4. The tree structure when both the tree edges and the ring edges are considered.

We now transform a problem instance in our model to a problem instance in the range model [16]. After we set up the tree structure, we set the contribution in the range model as follows. The contribution function of d_j towards the i-th ancestor of d_j is set to $C(v_j, i-1) - w(v_j)$ in our model. Recall that in our model $w(v)$ is the number of the requests that starts from v and $r(v)$ is the index of the server that will serve the data. The contribution from T'_{v_j} is represented by $C(v_j, i-1) - w(v_j)$, since it takes one step from d_j to v_j, and the workload from v_j should not be counted. On the other hand, we set the workload of v_j in the range limit model to be $w(v_j)$ and the range limit to be $r(v_j)$, to reflect the fact that requests from v_j must locate a server within $r(v_j)$ hops in our model.

After all the contribution functions for d_j and the workload on v_j are given, we can compute all the contribution functions for v_j in a bottom-up phase, using a dynamic programming approach described in [16]. During the process we also determine the locations to place the minimum number of replicas that can satisfy all requests. This will also determine the $m(T_v)$ in our model. If the dynamic programming places a replica at a dummy node d_j, it implies that we should place an additional replica at an arbitrary child of v_j.

Consider the example in Figure 3. We have a node v with five children; therefore, we need to consider five cases as described below. The five cases have v_1, v_2, v_3, v_4 and v_5 as the root of the subtree shown in Figure 4, respectively. After applying the optimal algorithm in the range model of [16] to these five cases respectively, we have five best server sets as follows. The server set for the first case includes five servers: v_1, v_2, v_4, v_5 and an arbitrary child of v_4. The server set for the second case includes five servers: v_1, v_2, v_3, v_4 and an arbitrary child of v_1. The server set for the third case includes five servers, v_1, v_3, v_4, an arbitrary child of v_1, and an arbitrary child of v_3. The server set for the fourth case

includes four servers, v_1, v_3, v_4 and an arbitrary child of v_1. The server set for the fifth case includes five servers, v_1, v_2, v_4, v_5 and an arbitrary child of v_4. Figure 5 shows the server set for the first case which has v_1 as the root of the subtree. The best server set for the case shown in Figure 6 is to assign v_4 as the root of the subtree because it uses only four servers, which is the smallest among the five cases.

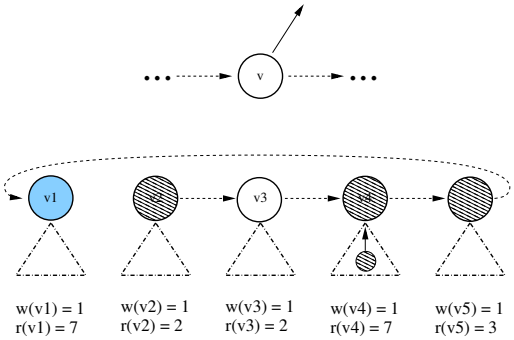

Fig. 5. The best server set for the case having v_1 as the root of the subtree structure illustrated in Figure 4.

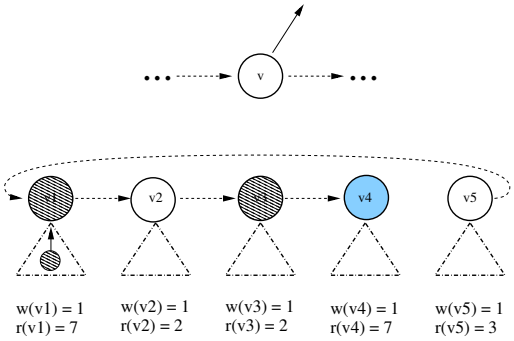

Fig. 6. The best server set for the case having v_4 as the root of the subtree structure illustrated in Figure 4.

5) The Final Answer: Let m_1 be the number of the minimum additional replicas for the first case where we assume no replica is placed at the children of v, and m_2 for the other case. The $m(T_v)$ is the minimum of m_1 and m_2, plus the summation of $m(T_{v_j})$ for all children.

$$m(T_v) = \min\{m_1, m_2\} + \sum_{1 \le j \le n} m(T_{v_j}). \qquad (6)$$

We choose the set of contribution function according to how we chose the m function. Note that if the m functions from two cases are equal, we will choose the second case, where there will be at least one replica among the children of v, so that the contribution of T'_v on v is zero. Figure 7 outlines the pseudo code for computing the m and C functions for node v and i-th node on the sibling path of v.

```
ComputePlaceRFunction(v, i):
if v is leaf {
   m(T_v) = 0
   if (i > r(v)) C(v, i) = ∞
   else C(v, i) = w(v)
} else {
   for each child c of v and 0 ≤ j ≤ i + n,
       call ComputePlaceRFunction(c, j).
   compute the m_1 and m_2 for the two cases of T_v.
   compute m(T_v) by equation (6).
   if the first case is optimal (m_1 < m_2) {
       if (i > r(v)) let C(v, i) = ∞
       else compute C(T_v, i) by equation (5).
   } else if the second case is optimal (m_2 ≤ m_1) {
       if (i > r(v)) C(v, i) = ∞
       else C(T_v, i) = w(v)
   }
}
```

Fig. 7. Computation of m and C functions

6) Time complexity analysis: We analyze the time complexity of our algorithm by focusing on the second case during the bottom-up computation of the C contribution function, which dominates the computation time. Let n be the number of children of v, then the computation of contribution function for v requires n^3 time. The reason is that we have to consider n cases where each case is a skewed tree of height n, and the computation for one level of the tree takes $O(n)$ time. Therefore the time complexity is the sum of the cubic of the number of children of every node. This number is bounded by the cubic of the total number of node, that is, $O(N^3)$ where N is the number of tree nodes in the data grid.

Theorem 1: There exists an algorithm that finds the optimal server set for the **PlaceR** problem in time $O(N^3)$, where N is the number of tree nodes in the data grid.

B. LoadBalance

We now derive an algorithm that solves the **LoadBalance** problem, i.e., we are given a number k and we want to place k replicas so that the maximum workload among them (including the implicit replica at the root) is minimized.

From the previous discussion we know that when given a tree T, a request set L with M requests, and a workload bound W, we are able to compute the minimum number of replicas so that all servers and the root have at most W workload. We can use this algorithm and a binary search to solve the **LoadBalance** problem as follows. We first "guess" a workload value B as the maximum workload on the replicas, and apply the previous algorithm for the **PlaceR** problem. If the number of replicas returned by the algorithm is greater than k, we should increase the workload B and try again; otherwise we should decrease the value of the workload B. We repeat this process until we find the *smallest* B such that the number of of replicas is k, and this is the server set we are looking for.

Now we analyze the time complexity. The maximum workload of a server must not be more than the number of the given priority lists; therefore the total workload is bounded by $|L| = M$. The number of rounds of binary search is therefore bounded by $O(\log |L|) = O(\log M)$. The total execution time of this binary search is therefore bounded by $O(N^3(\log M))$.

Theorem 2: There exists an algorithm that finds the optimal server set for **LoadBalance** in time $O(N^3(\log M))$, where N is the number of tree nodes in the data grid, and M is the number of the given requests.

V. Conclusion

This paper addresses data replica placement in Data Grid systems with the notion of priority lists. We propose the concept of priority list to deal with two issues that often arise in real-world Grid environments. First, a user may have limited authority in accessing the resources, and thus his/her data requests should be prohibited from accessing some of the sites. Second, a static request forwarding policy may not satisfy a data request with special requirements (e.g. quality of service requirement). Therefore, a suitable forwarding policy should allow individual requests to have varied forwarding paths. Our *priority lists*-based approach provides promising solution for this problem.

For replica placement, we focus on the Data Grid systems that can be modeled as sibling trees, since in a hierarchical grid, a request would usually visit all of the nodes managed by the same parent before it is forwarded to the parent. We propose two algorithms for the sibling tree model: **LoadBalance** and **PlaceR**. **LoadBalance** determines optimal locations for placing the replicas so that the workload among these replicas is balanced and all request are satisfied. **PlaceR** determines the minimum number of replicas when the maximum workload on each replica server is given.

We are working on several interesting related problems. For example, the replica placement problem for general graphs are usually NP-complete. We are currently developing effective heuristics for this class of problems. In addition, we are also interested in studying replica placement for Grids that can be modeled as planar graphs, for which efficient algorithms might exist. Finally, in the current hierarchical Data Grid model (e.g. sibling tree model), all the requests may go to the root if they cannot be served by a replica. This might cause network congestion, which is also an interesting goal for optimization.

a) **Acknowledgment:** This work is supported by the National Science Council under grant NSC96-2221-E-002-025 and NSC96-2628-E-001-004-MY3, the Ministry of Education under grant 95R0062-AE00-07, and the National Center for High-Performance Computing 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*, 2004, pp. 66–73.

[2] M. M. Bae and B. Bose, "Resource placement in torus-based networks," *IEEE Transactions on Computers*, vol. 46, no. 10, pp. 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*, no. 23, pp. 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*, vol. 40, no. 2, pp. 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*, 2003, pp. 120–126.

[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*, 2002, pp. 46–57.

[8] M. Deris, A. J.H., and H. 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] W. L. C. Grid, "http://lcg.web.cern.ch/lcg/."

[10] G. P. N. (GriphyN), "http://www.griphyn.org."

[11] S. Guha, R. Hassin, S. Khuller, and E. Or, "Capacitated vertex covering," *J. Algorithms*, vol. 48, no. 1, pp. 257–270, 2003.

[12] 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*, 2000, pp. 77–90.

[13] 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*, vol. 12, no. 6, pp. 628–637, June 2001.

[14] M. Korupolu, C. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching." *Journal of Algorithms*, vol. 38, no. 1, pp. 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*, 2002, pp. 378–383.

[16] P. Liu, , Y.-F. Lin, and J.-J. Wu, "Optimal placement of replicas in data grid environments with locality assurance." in *International Conference on Parallel and Distributed Systems*, 2006.

[17] 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*. Morgan Kaufmann PUblishers, 1999, ch. Data intensive computing.

[18] K. Ranganathan, A. Iamnitchi, and I. 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*, 2002, pp. 376–381.

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

[20] 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*, 2001, pp. 305–314.

[21] X. Tang and J. Xu, "Qos-aware replica placement for content distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, October 2005.

[22] 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*, vol. 7, no. 4, pp. 328–342, April 1996.

[23] O. Unger and I. Cidon, "Optimal content location in multicast based overlay networks with content updates," *World Wide Web*, vol. 7, no. 3, pp. 315–336, 2004.

[24] O. Wolfson and A. Milo, "The multicast policy and its relationship to replicated data placement," *ACM Trans. Database Syst.*, vol. 16, no. 1, pp. 181–205, 1991.