行政院國家科學委員會專題研究計畫 期中進度報告

分散式系統中為提升資料存取效率的複製策略(2/3)
期中進度報告(精簡版)

計 畫 類 別 ： 個別型
計 畫 編 號 ： NSC 95-2221-E-002-071-
執 行 期 間 ： 95 年 08 月 01 日至 96 年 07 月 31 日
執 行 單 位 ： 國立臺灣大學資訊工程學系暨研究所

計 畫 主 持 人 ： 劉邦鋒

報 告 附 件 ： 出席國際會議研究心得報告及發表論文

公 開 資 訊 ： 本計畫可公開查詢

中 華 民 國 96 年 11 月 03 日

附件一

行政院國家科學委員會補助專題研究計畫 □ 成果報告 ■期中進度報告

分散式系統中為提升資料存取效率的複製策略(2/3)

計畫類別：■個別型計畫　　□ 整合型計畫
計畫編號：NSC 95-2221-E-002-071
執行期間：　　　2006 年 8 月 1 日至 2007 年 7 月 31 日

計畫主持人：劉邦鋒
共同主持人：
計畫參與人員： 陳惠麟, 李鯀軒

成果報告類型(依經費核定清單規定繳交)：□精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：

中　華　民　國　　　　年　　　　月　　　　日

1 Introduction

Grid computing is an important mechanism for utilizing computing resources that are distributed in different geographical locations, but are organized to provide an integrated service. A grid system provides computing resources that enable users in different locations to utilize the CPU cycles of remote sites. In addition, users can access important data that is only available in certain locations, without the overheads of replicating it locally. These services are provided by an integrated grid service platform, which helps users access the resources easily and effectively. One class of grid computing, and the focus of this paper, is Data Grids, which provide geographically distributed storage resources for complex computational problems that require the evaluation and management of large amounts of data. For example, scientists working in the field of bioinformatics may need to access human genome databases in different remote locations. These databases hold tremendous amounts of data, so the cost of maintaining a local copy at each site that needs the data would be prohibitive. In addition, such databases are usually read-only, since they contain the input data for various applications, such as benchmarking, identification, and classification. 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 is to replicate the data in multiple locations so that a user can access the data from a server in his vicinity. It has been shown that data replication not only reduces access costs, but also increases data availability in many applications [7, 13, 12]. 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 [2, 5, 4, 6, 12, 14, 13, 15]. We believe that, to obtain maximum benefits from replication, a strategic placement of replicas is essential.

Although there has been much work on replica placement problem [10, 11, 17, 18, 20], very few of them concerns quality of service. A large part of these work concerns the average system performance, for example, to minimize the total accessing cost, or to minimize the total communication cost, etc. Although these metrics are important in the overall system performance, they cannot meet the individual requirement adequately. Grid computing infrastructure usually consists of various type of resources and the performance of these resources are quite diverse. Moreover, different sites may have different service quality requirements according to the system performance of the sites. Therefore, quality of service is an important factor in addition to overall system performance.

An early work by Tang and Xu [16] considered the quality of service in addition to minimize the storage and update cost. The distance between two nodes is used as a metric for quality assurance. A request must be answered by a server within the distance specified by the request. Every request knows the nearest server that has the replica and the request takes the shortest path to reach the server. Their goal has been to find a replica placement that satisfies all requests without violating any range constraint, and minimize the update and storage cost at the same time. They show that this QoS-aware replica placement problem is NP-Complete for general graphs, and provide two heuristic algorithms – l-`Greedy-Insert` and l-`Greedy-Delete`, for general graphs. A dynamic programming solution is given for tree topology [16].

In this paper, we study the QoS-aware replica placement problem for general graphs; moreover, we take the workload capacity limit of each replica server and the access cost of each data request into consideration. When a data request is dispatched to a overloaded server, it will not get a response in time. Therefore, we believe that quality of service and workload capacity should be considered simultaneously for quality assurance. In addition, the data access cost has a profound influence on the overall system performance, therefore the access cost must be taken into account to improve system performance. In our model, each request must be serviced by a replica server within its quality requirement *and* without violating the capacity limits of the replica server. We provide two heuristic algorithms to decide the positions of the replicas to minimize the sum of update, storage and access costs, and satisfy the quality requirements specified by the user and the capacity limit that each replica server can service. Our algorithm computes near-optimal solutions efficiently, so that it can be deployed in various realistic environments.

2 Related Works

Optimal replica placement problem has been studied extensively in the literature. The same problem has different names in different research areas. For example, it is refereed to as p-median problem in operations research, or database location problem on Internet and file allocation problem in computer science. Wolfson and Milo [20] proved that replica placement problem is NP-Complete for general graphs when read and update cost are simultaneously considered. They also provide optimal solutions for special topologies, including complete graph, tree, and ring. Tu et al. [17] study the secure data placement problem in the same model and provide a heuristic algorithm for general graphs. Krick et al. [11] consider read, update and storage cost simultaneously in general graph, and provide an polynomial time approximation algorithm that has a constant competitive ratio. They also provide an optimal solution for tree topology in the same paper. Kalpakis, Dasgupta and Wolfson [10] consider read, update and storage cost under tree topology. Their algorithm could cope with the situations even when servers have capacity limits. They describe an $O(n^3k^2)$ dynamic programming algorithm for k replicas placed in n incapacitated servers, and an $O(n^3k^2\wedge_{max}^2)$ algorithm for capacitated servers, where $\wedge_{max}$ denotes the maximum capacity among all servers. Unger and Cidon [18] provide a more efficient algorithm to find the optimal placement under similar model, with only $O(n^2)$ time, where n is the number of servers. However, the algorithm in [18] cannot deal with server capacity limits. There are other algorithms that provide optimal solutions under simpler models for tree topology [9, 3].

Although there has been a lot of work studying the optimal replica placement problem, very few of them concerns quality of service. The goal in these efforts is usually to minimize the total replication cost. The replication cost may contain read, update and storage cost, depending on the system model. The objective has usually been to improve the average system performance, without any quality-of-service guarantees. An early effort by Tang and Xu [16] suggested a QoS-aware replica placement problem to cope with the quality-of-service issues. Every edge uses the distance between the two end-points as a cost function. The distance between two nodes is used as a metric for quality assurance. A request must be answered by a server that is within the distance specified by the request. Every request knows the nearest server that has the replica and the request takes the shortest path to reach the server. Their goal has been to find a replica placement that satisfies all requests without violating any range constraint, and minimizes the update and storage cost at the same time. They show that QoS-aware replica placement problem is NP-Complete for general graphs, and provide two heuristic algorithms, called l-`Greedy-Insert` and l-`Greedy-Delete`, for general graph, and a dynamic programming solution for tree topology.

l-`Greedy-Insert` starts with an empty replication set R, and inserts replicas into R until all servers' QoS requirements are satisfied. In the first step, the algorithm selects $(l+1)$ replicas that maximize the *normalized benefits* among all possible locations. Normalized benefits is defined as the increased number of satisfied servers divided by the increased replication cost due to the selection. In each step, we examine all possible replacement, each of them replaces l replicas with some $(l+1)$ replicas, and choose the one that maximizes the normalized benefits. Note that the removed replicas and the inserted replicas can overlap.

l-`Greedy-Delete` works the opposite way as the l-`Greedy-Insert`. We begin with having a replica in every node, then it deletes replicas whose deletion maximizes the replication cost reduction until there is no replica that can be deleted. In the first step, l-`Greedy-Delete` removes the $(l+1)$ replicas whose deletion maximizes replication cost reduction without violating the QoS requirements. In each subsequent step, the algorithm examines all possibilities of replacing $(l+1)$ replicas with l replicas without violating QoS requirements, and chooses the one that maximizes replication cost reduction. We repeat the process until there is no possible alternative left.

The time complexity of l-`Greedy-Insert` and l-`Greedy-Delete` is $O(|V|^3)$ for $l = 0$ and $O(|V|^{2l+2})$ for any $l > 0$ [16]. The time complexity for the $l = 0$ case is due to shortest path computation. There is a trade-off between the time complexity and the quality of solution on l value. Although the time complexity is a polynomial function of the number of nodes, the execution time of these two algorithms are very slow in practice even when $l = 1$.

Since l-`Greedy-Insert` starts by inserting replicas into a empty replica set, and l-`Greedy-Delete` starts by deleting replicas from a full replica set, the execution time of these two algorithms depends heavily on the number of replicas in the optimal solution. If the optimal solution has very few replicas, l-`Greedy-Insert` becomes more efficient than l-`Greedy-Delete`. On the other hand, l-`Greedy-Delete` is much more efficient when the optimal solution contains a lot of replicas.

Won, Indranil and Klara proposed a simpler formulation about QoS-aware replica placement problem [8]. The model did not consider update cost and assumed that each server has identical storage cost. The goal was to minimize the number of replicas in the system. They gave a proof of NP-Completeness for this problem, which is a variation of set covering. Let A be the all-to-all shortest path matrix and entry (i, j) of A denotes the shortest path distance between server i and server j. Let B be another matrix and the entries in the i-th row indicate quality of service requirement of server i. We then construct another matrix C according to $A - B$. If an entry of $A - B$ is less than or equals to 0, we set set the corresponding entry of C to 1. Otherwise, we set the entry to 0. The non-zero elements of the j-th column of C represents the servers that are covered by server j. If we find a set of columns that cover every row in matrix C, we find a replica placement that satisfies all requests within quality of service requirement.

Won, Indranil and Klara proposed a simpler and quicker algorithm to find a reasonable good solution for this problem. Every iteration in the algorithm, they select the column j (server j) that covers most rows that not yet covered so far. This *Greedy MSC* (Greedy Minimum Set Covering) is compared with our methods in our simulation testing.

Our model differs from the model in [16] because it considers not only site construction, update overheads, quality of service, both the *capacity constraint* and the *access costs*. The capacity constraint of each replica server is an important factor in the requsets response time, and the access cost of each request has a great influence on system performance. We believe that these two factors should be taken into consideration, along with site construction, update overheads, and quality of service. In this paper, we propose two heuristic algorithms to find near optimal solutions, while all the constraints, including capacity constraint and access costs, are considered.

3 System Model

This section describes our system model. The network is represented by an undirected graph $G = (V, E)$, where V is the set of servers, and $E \subseteq V \times V$ denotes the set of network links among the servers. Each link $(u, v) \in E$ is associated with a cost $d(u, v)$ that denotes the communication cost of the link. We assume that the graph is connected, so that one server can connect to any other server via a path. We define the communication cost of a path as the sum of the communication cost of the links along the path. Because we assume that a server knows where to find the replica that services it, we define $d(u, v)$ between two servers u, v to be the communication cost of the *shortest* path between them. Every server u has a storage cost, $\mathcal{S}(u)$, that denotes the cost to put a replica on server u. The storage cost on different nodes may be different. Figure 1 is an example of our model. The numbers in the circles are server indices between 0 and $n - 1$, where n is the total number of servers. The number next to a server is its storage cost. The number on a link is the communication cost of the link.

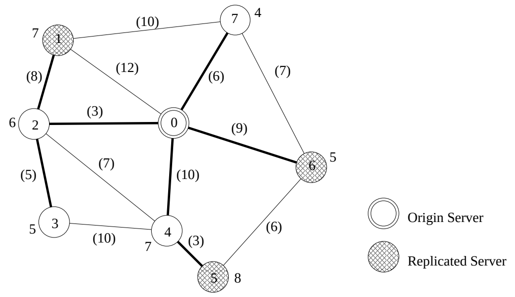

Figure 1: An example of data replication in connected network.

Each server in the network services multiple clients, although we do not place clients into the network graph. A client sends its requests to its associated server, then the server processes the requests. If the client's requests can be served by the server, i.e., the local server has the requested data, the requests will be processed locally. Otherwise, the requests will be directed to a server that has the replica. As a result, we assume that all requests are issued from the servers and there are only servers in the network graph. In addition, because the communication cost from the clients to servers does not affect the replication decision, we ignore the communication cost from clients to servers.

There is a special server r, called *origin server*, in the network graph. Without lose of generality, we assume that server 0 is the origin server. Initially only the origin server has the data. A *replica server* is a server that has a copy of the original data.

A *replication strategy* has two parts: a *replication server set* $R \subseteq V - \{r\}$, and a *service set function* SS. For each server $u \in R \cup \{r\}$, we define a *service set* function $SS(u)$ to be the set of servers that u services. We assume that each server goes to only one server in $R \cup \{r\}$ for service, therefore for two distinct servers $u, v \in R \cup \{r\}$, the service sets of u and v are disjoint, i.e., $SS(u) \cap SS(v) = \emptyset$. In other words the service set function SS is a *partition* of all servers in V among $R \cup \{r\}$

3.1 Replication Cost

We use *replication cost* to evaluate replication strategies. The replication cost $T(R)$ of a replication strategy is defined as the sum of the *storage cost* $S(R)$, *update cost* $U(R)$, and *access cost* $A(R)$.

$$T(R) = S(R) + U(R) + A(R) \tag{1}$$

Storage cost The storage cost of a replication strategy is the sum of all storage cost of the replica servers in the replication server set R. Recall that $\mathcal{S}(v)$ is the storage cost to replica a data at server v.

$$S(R) = \sum_{v \in R} \mathcal{S}(v) \tag{2}$$

Update cost In order to maintain data consistency, the origin server r issues update requests to every replica server. The update frequency μ denotes the number of update requests issued by r per time period. We assume that there is an *update distribution tree* T, which connects all the servers in the network. For example, in our experiments, we use a shortest path tree rooted at the origin server as the update distribution tree. As in Figure 1, we use bold lines to represent the edges of the shortest path tree. The origin server r multicasts update requests through links on this tree until all the replica servers in R receive the update requests. Every node receives update requests from its parent and relays these requests to its children according to the update distribution tree.

Given the network, the update distribution tree, the update frequency μ, the update cost of a replication servers R is defined as follows. Let $p(v)$ be the parent of node v in the update distribution tree, and T_v be the subtree rooted at node v. If $T_v \cap R \neq \emptyset$, the link $(v, p(v))$ participates the update multicast. As a result, the update cost is the sum of the communication costs from these links $(v, p(v))$. For example, in Figure 1 if the update rate is 1 and the replication servers R is $\{1, 5, 6\}$, then the update cost is $11 + 13 + 9 = 33$.

$$U(R) = \mu \times \sum_{v \neq r,\ T_v \cap R \neq \emptyset} d(v, p(v)) \tag{3}$$

Access cost The access cost of a replication strategy is defined as the following. Each server v has to communicate with a replica server u when it wishes to access the data from u, where $v \in SS(u)$. The access cost of a replication strategy is the sum of the communication cost that each server v accesses the data from its assigned replica server according to the service set function SS, as in the following equation.

$$A(R) = \sum_{u \in R \cup \{r\}} \sum_{v \in SS(u)} d(u, v) \tag{4}$$

3.2 Service Quality Requirement

Every server u has a *service quality requirement* $\mathcal{Q}(u)$. The requirement mandates that all requests generated by u will be serviced by a server within $\mathcal{Q}(u)$ communication cost. If requests from server u is serviced by a replica server within distance $\mathcal{Q}(u)$ from u, server u is *satisfied*.

3.3 Workload Capacity Constraint

Each server u has a workload $\mathcal{W}(u)$ and workload capacity constraint $\mathcal{C}(u)$. The workload $\mathcal{W}(u)$ of a server is defined as the number of requests generated by server u. For each server u, when we put a replica on u, it has a workload capacity constraint, $\mathcal{C}(u)$, that denotes the amount of data requests that the replica server u can handle. The origin server also has its workload capacity constraint $\mathcal{C}(r)$. The workload and workload capacity constraint on different server may be different. If the total workload that a server $u \in R \cup \{r\}$ services is greater than its capacity constraint, i.e., $\sum_{v \in SS(u)} \mathcal{W}(v) > \mathcal{C}(u)$, the server u is *overloaded.*

3.4 QoS-aware Replica Placement With Capacity Constraint

A replication strategy is *feasible* if all servers are satisfied, and none of the server $u \in R \cup \{r\}$ is overloaded. The problem of QoS-aware replica placement with capacity constraint is to find a feasible replication server set R and determine the service set function $SS(u)$ for each server $u \in R \cup \{r\}$, such that the replciation cost in Equation (1) is minimized.

4 Heuristic Algorithms

In this section we first propose two heuristic algorithms – **Greedy-Remove** and **Greedy-Add** for QoS-aware replica placement with capacity constraint problem. Then, we analyze the time complexity of these two algorithms.

4.1 QoS Satisfying Set

We define a *QoS satisfying set* $SAT(u)$ of a server u to be the set of servers from which u is located within their QoS distance $\mathcal{Q}$. That means should u become a available replica server, it is able to satisfy those nodes in $SAT(u)$. Formally we have Equation 5.

$$SAT(u) = \{v | d(u, v) \leq \mathcal{Q}(v)\} \tag{5}$$

Each server has its own QoS satisfying set. If there is a replica on server u, each server $v \in SAT(u)$ may be satisfied by u, if u will not be overloaded by doing so . For a feasible replication strategy, the service set of each server $u \in R \cup \{r\}$ must be a subset of its QoS satisfying set $SAT(u)$, that is, $SS(u) \subseteq SAT(u)$ for all $u \in R \cup \{r\}$.

4.2 Greedy Remove

The algorithm **Greedy-Remove** starts with having a replica on *every* server. This replication strategy is feasible since every server can serve itself locally so any QoS constraint is satisfied. Therefore the service set of each server $u \in R \cup \{r\}$ has only itself. **Greedy-Remove** then repeatedly adjusts the service sets SS of a pair of replica servers and try to remove replicas in order to reduce the replication cost (Equation (1)). While removing replicas, **Greedy-Remove** must simultaneously maintain the feasibility of the replication strategy.

We now describe our **Greedy-Remove** method in details. The **Greedy-Remove** method works in rounds. Initially we put a replica on every server in $V - \{r\}$, so the replication server set R is $V - \{r\}$. For each iteration **Greedy-Remove** examines each pair of servers u, v in $R \cup \{r\}$ and computes the cost reduction function $\mathcal{R}(u, v)$. Then **Greedy-Remove** selects the maximum $\mathcal{R}(u, v)$ and adjust the service sets of u and v accordingly. **Greedy-Remove** repeats this process until it is impossible to reduce the total replication cost.

4.2.1 Time Complexity Analysis

We now analyze the time complexity of **Greedy-Remove**. The first part of the costs is a preprocessing to find a shortest path between any two servers. That is, we must build all-pair shortest path to check if one server is within the QoS requirement of another. This takes $O(|V|^3)$ time to calculate.

Given the servers u and v, it takes $O(|V|)$ to find out the cost reduction of moving all servers from v to u for the first case. For the second case it takes $O(|V| log|V|)$ time to sort the servers, then examine them one by one and move them from u to v if its distance to v is longer than the distance to u.

We first compute the cost reduction function $\mathcal{R}$ of each pair of replica servers. This takes $O(|V|^3 log|V|)$ time.

In each iteration, we choose the largest cost reduction $\mathcal{R}$. If the corresponding case is the first case, we need to consider two situations. The first is that there is no replica in the subtree rooted at v after removing the replica from v. We have to recompute the cost reduction function of v's parent replica server with the other replica servers. Because its reduced update cost in rm function has been changed. The other situation is that there is only one replica in the subtree T_v after taking off the replica from v. We have to recompute the cost reduction function of this replica server with the other replica servers. This is because the reduced update cost of this replica server is increased. Hence if the corresponding case is the first case, it needs $O(|V|^2)$ time. Otherwise, if the corresponding case is the second case, we only need to recompute the cost reduction function of the two replica servers that we selected with the other replica servers. This takes $O(|V|^2 log|V|)$. Thus, each iteration needs $O(|V|^2 log|V|)$ time.

In each iteration we remove a replica from a server or move at least one server from one service set to another.

The number of replica removal is at most $|V|$ since a replica can be removed at most once.

In addition, each server without placing the replica can be moved at most $|V|$ times because it can only be moved from a service set to another once, and there are $|V|$ servers in the network system.

Therefore, there are $O(|V|^2)$ iterations. As a result, the time complexity of **Greedy-Remove** is $O(|V|^3 + |V|^3 log|V| + |V|^2 \cdot |V|^2 log|V|) = O(|V|^4 log|V|)$.

4.3 Greedy Add

The **Greedy-Add** algorithm works the opposite way as **Greedy-Remove** does. It begins with an empty replication server set R, and add replicas to R one at a time.

In the second stage, **Greedy-Add** repeatedly adds replica in order to decrease the access cost. This stage works also in iterations. In each iteration, **Greedy-Add** examines all servers $u \in V - R \cup \{r\}$, and try to place a replica on a server u to determines whether this will reduce the replication cost.

When we put the replica on a server $u \in V - R \cup \{r\}$, u has to serve itself and u must be put into $SS(u)$. Then we start selecting servers from $SAT(u)$ and move them to $SS(u)$. We select servers $v \in SAT(u)$ whose communication cost $(d(v, u))$ is less than the communication to its original server $(d(v, rep(v)))$. The selection starts from the server v with the largest largest $d(v, rep(v)) - d(v, u)$, until there is no available capacity on u, or no server left in $SAT(u)$. We then recompute the replication cost. After trying all servers in $V - R \cup \{r\}$, **Greedy-Add** puts the next replica on a server that reduces the replication cost most. **Greedy-Add** repeats the process until it is impossible to reduce the replication cost.

4.3.1 Time Complexity Analysis

We analyze the time complexity of **Greedy-Add**. Similar to the analysis of **Greedy-Remove**, the first part of the costs is a preprocessing to find a shortest path between any two servers, which takes $O(|V|^3)$ time to calculate.

In the first stage, **Greedy-Add** inserts the replicas into the network iteratively until all servers are served. In each iteration, **Greedy-Add** examines $O(|V|)$ servers. If we put the replica on a server, it takes $O(|V|)$ time to determine the service set and to compute the normalized benefit. Thus each iteration takes $O(|V|^2)$ time. In each iteration, **Greedy-Add** puts the replica on a server and there are at most $|V|$ servers in the network. Therefore, in the first stage, **Greedy-Add** requires $O(|V|^3)$ time to determine a feasible replication strategy.

In the second stages, **Greedy-Add** inserts the replicas into the network to reduce the replication cost until further reduction in replication cost is not possible. In each iteration, **Greedy-Add** examines $O(|V|)$ servers. It takes $O(|V| log|V|)$ time to determine the service set and $O(|V|)$ time to recompute the replication cost. Since there are $|V|$ servers in the network, the second stage of **Greedy-Add** takes $O(|V|^3 log|V|)$ time to finish. As a result, the overall complexity of **Greedy-Add** is $O(|V|^3 log|V|)$.

4.4 Random

We randomly insert the replica into the network until all servers are served. The servers in a QoS satisfying set $SAT(u)$ are sorted according to their communication cost to u. When we put the replica on a server u, u has to serve itself first. Then we select servers from $SAT(u)$ to form $SS(u)$. The selection starts from the first server v in $SAT(u)$ that has not been served by any server in $R \cup \{r\}$ until there is no workload capacity available on u, or every server in $SAT(u)$ have already been served by $R \cup \{r\}$.

5 Conclusion

Data replication is an important technique to speed up data access in Data Grid. Grid computing infrastructure usually consists of various type of resources and the performance of these resources are quite diverse. So to provide quality assurance for different data access requirements is more and more important. This replica placement problem become more complex when the storage for replica on servers is limited. We believe that quality of service and workload capacity should be considered simultaneously for quality assurance, and the access cost must also be taken into account for system performance.

In this paper, we consider QoS requirement, workload capacity restriction and access cost on replica placement problems. We believe that all the key issuers, including storage cost, quality of service, server capacity constraint, access costs, and update costs should be consider. Our proposed algorithms consider all these key issues, and is very simple and easy to adapt to variant environments. Experiment results indicate that **Greedy-Remove** and **Greedy-Add** can find near-optimal solutions in all parameter combinations.

Acknowledge The authors thank the support from National Science Council, under grant NSC 95-2221-E-002-071.

References

[1] GT Internetwork Topology Models (GT-ITM), 2000. http://www-static.cc.gatech.edu/projects/gtitm/.

[2] 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.

[3] Israel Cidon, Shay Kutten, and Ran Soffer. Optimal allocation of electronic content. In *INFOCOM*, pages 1773–1780, 2001.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Won J. Jeon, Indranil Gupta, and Klara Nahrstedt. Qos-aware object replication in overlay networks. 2005.

[9] Xiaohua Jia, Deying Li, Xiao-Dong Hu, and Ding-Zhu Du. Placement of read-write web proxies in the internet. In *ICDCS*, pages 687–690, 2001.

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

[11] Christof Krick, Harald Räcke, and Matthias Westermann. Approximation algorithms for data management in networks. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 237–246, New York, NY, USA, 2001. ACM Press.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Xueyan Tang and Jianliang Xu. Qos-aware replica placement for content distribution. *IEEE Trans. Parallel Distrib. Syst.*, 16(10):921–932, 2005. Member-Xueyan Tang and Member-Jianliang Xu.

[17] Manghui Tu, Peng Li, Qingkai Ma, I-Ling Yen, and Farokh B. Bastani. On the optimal placement of secure data objects over internet. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 14, Washington, DC, USA, 2005. IEEE Computer Society.

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

[19] Bernard M. Waxman. Routing of multipoint connections. pages 347–352, 1991.

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

出席國際學術會議心得報告

計畫編號	NSC 95-2221-E-002-071
計畫名稱	分散式系統中為提升資料存取效率的複製策略(2/3)
出國人員姓名 服務機關及職稱	劉邦鋒，台大資訊系教授
會議時間地點	Barcelona, Spain, September 28th-29th 2006
會議名稱	7th IEEE/ACM International Conference on Grid Computing
發表論文題目	A QoS-Aware Heuristic Algorithm for Replica Placement

一、參加會議經過

This a report about attending the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, September 28th-29th 2006. This conference is probably the only grid computing conference sponsored by both ACM and IEEE. It is an annual international meeting that brings together a community of researchers, developers, practitioners, and users involved with Grid technology. This year the acceptance rate is 18%.

I presented my paper in session 3B Data Resource Allocation during the first day. The title of my paper is “A QoS-Aware Heuristic Algorithm for Replica Placement”. This is a joint work with Dr. Jan-Jan Wu from the institute of Information Science, Academia Sinica, and my student Hsing-Kai.Wang from National Taiwan University. Several interesting questions were raised after my talk and I think future extension is possible.

二、與會心得

One of the most impressive keynote speech was delivered by Dr. Malcom Atkinson, who described the current status of grid computing environment in U.K. The current status of e-Science in U.K. is also covered in his talk. I personally feel that Europeans have put much more resources in grid computing than Americans do. I also believe that this is money well spent, since the performance index of European e-science is measured in the number of Nobel’s prizes and the number of articles in Science and Nature. Grid computing provides a virtual platform so that researchers in different disciplines can work seamlessly together. I think this is the most important goal in grid computing.