

# Optimizing Server Placement in Hierarchical Grid Environments

Chien-Min Wang<sup>1</sup>, Chun-Chen Hsu<sup>1</sup>, Pangfeng Liu<sup>2</sup>, Hsi-Min Chen<sup>3</sup>, and Jan-Jan Wu<sup>1</sup>

<sup>1</sup> Institute of Information Science, Academia Sinica, Taipei, Taiwan  
{cmwang, tk, wuj}@iis.sinica.edu.tw

<sup>2</sup> Department of Computer Science and Information Engineering,  
National Taiwan University, Taipei, Taiwan  
pangfeng@csie.ntu.edu.tw

<sup>3</sup> Department of Computer Science and Information Engineering,  
National Central University, Taoyuan, Taiwan  
seeme@selab.csie.ncu.edu.tw

**Abstract.** In this paper, we address some problems related to server placement in Grid environments. Given a hierarchical network with requests from clients and constraints on server capability, the minimum server placement problem attempts to place the minimum number of servers that satisfy clients requests. Instead of using a heuristic approach, we propose an optimal algorithm based on dynamic programming to solve the problem. We also consider the balanced server placement problem, which tries to place a given number of servers appropriately so that their workloads are as balanced as possible. We prove that an optimal server placement can be achieved by combining the above algorithm with a binary search of workloads. We extend this approach to deal with constraints on network capability. The simulation results clearly show an improvement in the number of servers and the maximum workload. Furthermore, as the maximum workload is reduced, the waiting times are reduced accordingly.

## 1 Introduction

Grid technologies, which enable scientific applications to utilize a wide variety of distributed computing and data resources, classified into two categories: Computing Grids and Data Grids [1, 2]. A Data Grid is a distributed storage infrastructure that integrates distributed, independently managed data resources. It addresses the problems of storage and data management, data transfers and data access optimization, while maintaining high reliability and availability of the data. In recent years, a number of Data Grid projects have emerged in various disciplines, for instance, EU Data Grid [3], PPDG [4], iVDGL [5], GriPhyN [6] and BIRN [7].

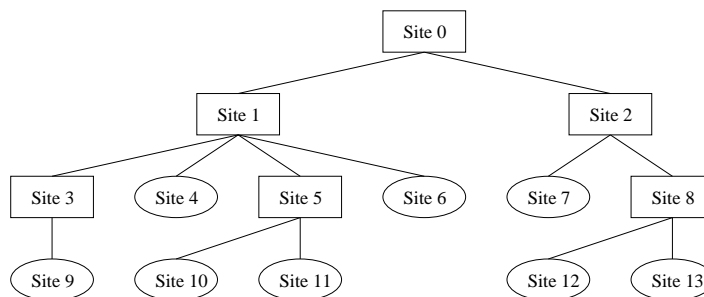
One way of solving the data access optimization problems is to distribute multiple copies of a file across different server sites in the grid system. It has been

shown that file replication can improve the performance of the applications [8–11]. The existing works focus on how to distribute the file replicas in a data grid in order to optimize different criteria such as I/O operation costs [11], response time and bandwidth consumption [9].

In this paper, we focus on some server placement problems in Data Grid environments. Given a hierarchical network with requests from clients and constraints on server capability, the solution to minimum server placement problem attempts to place the minimum number of servers that can satisfy clients requests. Instead of using a heuristic approach, we propose an optimal algorithm based on dynamic programming to solve this problem. We also consider the balanced server placement problem, which tries to place a given number of servers appropriately so that their workloads are as balanced as possible. We prove that an optimal server placement can be achieved by combining the above algorithm with a binary search on workloads. We extend this approach to deal with constraints on network capability. The experiment results clearly show the improvement in the number of servers and the maximum workload. Furthermore, as the maximum workload is reduced, waiting times are also reduced.

## 2 Background

In this paper, we use a hierarchical Grid model, one of the most common architectures in current use [8, 9, 12–14]. Consider Fig. 1 as an example. Leaf nodes represent client sites that send out I/O requests. The root node is assumed to be the I/O server that stores the master copies of all files. Without loss of generality, we assume that root node is the site 0. Intermediate nodes can be either routers for network communications or I/O servers that store file replicas. Edges represent communication channels between nodes. We further assume that, initially, only one copy (i.e., the master copy) of a file exists at the root site, as in [9, 13].



**Fig. 1.** The hierarchical Grid model

Associated with each client site  $i$ , there is a parameter  $r_i$  that represents the arrival rate of read requests for client site  $i$ . A data request travels upward from

a client site and passes through routers until it reach an I/O server on the path. Upon receiving the request, the I/O server sends data back to the client site if it owns a copy of the requested file. Otherwise, it forwards the request to its parent server. This process continues up the hierarchy recursively until a node that has the requested file is encountered or the root node is reached. The root server might update the contents of the file. For each update, corresponding update requests are sent to the other I/O servers to maintain file consistency. Let  $u$  be the arrival rate of update requests from the root server.

Associated with each server site  $j$ , there is a parameter  $\lambda_j$  that represents the arrival rate of I/O requests;  $\lambda_j$  can be written as:  $\lambda_j = \sum_{i \in C_j} r_i + u$ , where  $C_j$  is the set of clients served by server site  $j$ . The first term represents the read requests generated by clients in  $C_j$ . The second term denotes the update requests that will be sent to server site  $j$ . We can further generalize this model so that each edge has its own connectivity bandwidth constraints.

In the absence of file replicas, all I/O requests must be served by the roots node. However, the request arrival rate is usually much higher than the service rate of the root node so that clients have to wait indefinitely for service. By placing I/O servers between client sites and the root node, some of I/O requests can be served by these I/O servers thereby alleviating the workload on the root node. According to Queueing Theory, the workload of I/O servers is the dominant factor in the waiting time of I/O requests. Therefore, to benefit from file replicas, it is important to place I/O servers at appropriate locations in a hierarchical Grid system.

### 3 The minimum server placement problem

I/O requests generated by client sites and data transfer requests served by server sites can be modeled as queueing systems. According to Queueing Theory, the queue length and the waiting time of a queueing system will eventually reach infinity if the arrival rate of data is greater than the service rate. Hence, there is a hard constraint on the arrival rate of each I/O server in a Grid system. File replicas present a natural solution to this problem. By placing the replicas with more I/O servers, it is possible to share I/O requests along servers and balance their workload. However, it is quite expensive to set up I/O servers in a Grid system, as having more servers usually lead to lower utilization, which means a waste of the systems resources and increased maintenance costs. Therefore, our first problem is to place the minimum number of I/O servers that will balance the workload of I/O requests.

**Definition 1.** Given the network topology, request arrival rates and I/O service rates, the *minimum server placement problem* tries to place the minimum number of I/O servers such that the arrival rate of requests that reach each I/O server is less than its service rate

To solve this problem, we intuitively employ a greedy method, similar to that in [12], by placing I/O servers one by one until all the servers including the root

server meet their constraints. Although this algorithm is rather fast and easy to implement, we found that it did not always generate the minimum number of servers in our experiments. Therefore, instead of employing a heuristic approach, we try to find an optimal algorithm based on the dynamic programming approach as shown in the remainder of this section.

**Definition 2.** Let  $L(i, m)$  be the minimum arrival rate of *leakage requests* that pass through node  $i$  when at most  $m$  servers are placed in the sub-tree rooted at node  $i$ , and the arrival rate of requests that reach each I/O server is less than its service rate.

Leakage requests that pass through node  $i$  are requests generated by leaf nodes in the sub-tree rooted at node  $i$ , but not served by the I/O servers in that sub-tree. Such requests must be serviced by an I/O server above node  $i$  in the hierarchy. Hence, it is desirable to minimize the arrival rate of these leakage requests. Depending on the server placement, the arrival rate of the leakage requests may change.  $L(i, m)$  represents the minimum arrival rate of leakage requests among all possible placements of at most  $m$  servers. Let  $n$  be the number of nodes in the Grid system. Based on the following theorems, such a minimum arrival rate can be computed in a recursive manner.

**Theorem 1.**  $L(i, m + 1) \leq L(i, m)$  for any node  $i$  and  $m \geq 0$ .

**Theorem 2.** If node  $i$  is a leaf node, then  $L(i, m) = \lambda_j$  for  $0 \leq m \leq n$ .

*Proof.* Since a leaf node cannot be an I/O server, all I/O requests generated by a client site will travel up the tree to the leaf nodes parent. By Definition,  $L(i, m) = \lambda_j$  for  $0 \leq m \leq n$ .  $\square$

**Theorem 3.** For an intermediate node  $i$  with two child nodes  $j$  and  $k$ , we can derive:

$$L(i, m) = 0 \text{ if } \min_{0 \leq r \leq m-1} \{L(j, r) + L(k, m - r - 1)\} \leq \mu_i$$

$$L(i, m) = \min_{0 \leq r \leq m} \{L(j, r) + L(k, m - r)\}, \text{ otherwise}$$

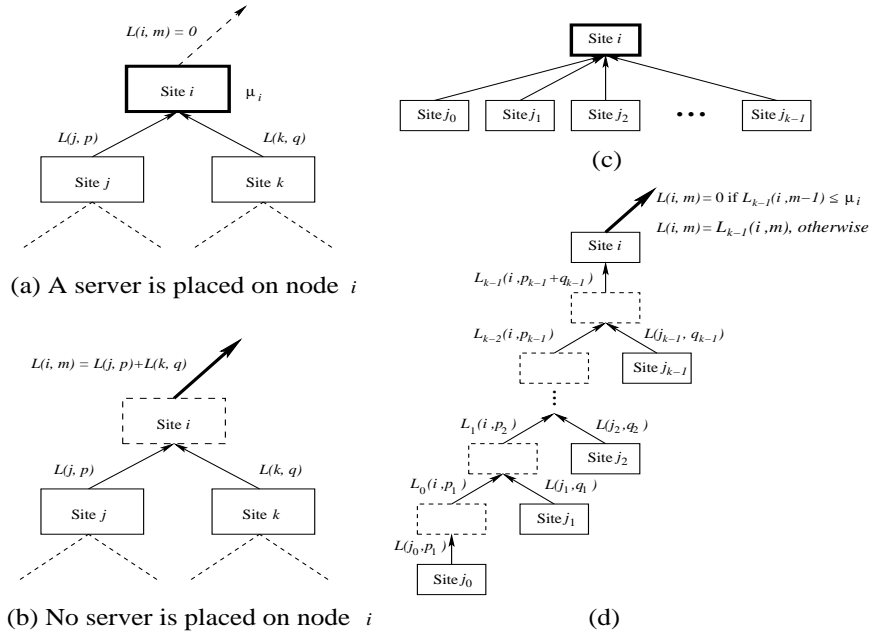
*Proof.* Case 1: A server is placed on node  $i$ . Consequently, at most,  $m - 1$  servers are placed on sub-trees rooted at node  $j$  and node  $k$ . This happen if and only if  $\min_{0 \leq r \leq m-1} \{L(j, r) + L(k, m - r - 1)\} \leq \mu_i$ . The “if” part can be proved as follows. Suppose that the minimum can be obtained when there are  $p$  servers on the sub-tree rooted at node  $j$  and  $q$  servers on the sub-tree rooted at node  $k$  as shown in Fig. 2(a). By Definition, the minimum arrival rate of leakage requests that pass through node  $j$  and node  $k$  will be  $L(j, p)$  and  $L(k, q)$  respectively. Since node  $i$  has only two child nodes,  $j$  and  $k$ , the arrival rate of I/O requests that reach node  $i$  must be the sum  $L(j, p) + L(k, q)$ . Accordingly, we can derive:

$$L(j, p) + L(k, q) = \min_{0 \leq r \leq m-1} \{L(j, r) + L(k, m - r - 1)\} \leq \mu_i$$

Hence, a server can be placed on node  $i$ . In this case,  $L(i, m) = 0$  and must be optimal. The “only if” part can be proved similarly. Suppose that, in an optimal

server placement, there are  $p$  servers on the sub-tree rooted at node  $j$  and  $q$  servers on the sub-tree rooted at node  $k$ . Obviously, we have the inequalities  $0 \leq p, q \leq m - 1$  and  $p + q \leq m - 1$ . Since node  $i$  has only two child nodes,  $j$  and  $k$ , the arrival rate of I/O requests that reach node  $i$  must be the sum  $L(j, p) + L(k, q)$  and must meet the constraint  $L(j, p) + L(k, q) \leq \mu_i$ . According to Theorem 1, we can derive:  $\mu_i \geq L(j, p) + L(k, q) \geq L(j, p) + L(k, m - 1 - p) \geq \min_{0 \leq r \leq m-1} \{L(j, r) + L(k, m - r - 1)\}$ . This completes the proof of case 1.

Case 2: No server is placed on node  $i$ . Consequently, at most  $m$  servers are placed on sub-trees rooted at nodes,  $j$  and  $k$ . Suppose that, in an optimal server placement, there are  $p$  servers on the sub-tree rooted at node  $j$  and  $q$  servers on the sub-tree rooted at node  $k$ , as shown in Fig. 2(b). Obviously, we have the inequalities  $0 \leq p, q \leq m$  and  $p + q \leq m$ . Since node  $i$  has only two child nodes,  $j$  and  $k$ , the arrival rate of I/O requests that reach and pass through node  $i$  can be computed as:  $L(i, m) = L(j, p) + L(k, q) \geq L(j, p) + L(k, m - p) \geq \min_{0 \leq r \leq m} \{L(j, r) + L(k, m - r)\}$ . According to above the assumption, this is an optimal server placement. Hence, all the equalities must hold. This completes the proof of case 2.  $\square$



**Fig. 2.** (a). (b) Illustrate two possible server placements on node  $i$ . (c). (d) Illustrate the basic concept of Theorem 4.

**Theorem 4.** For an intermediate node  $i$  with  $k$  child nodes  $j_0, j_1, \dots, j_{k-1}$ , the minimum arrival rate of leakage requests that pass through node  $i$  can be computed iteratively as follows:

$$L_0(i, m) = L(j_0, m),$$

$$L_q(i, m) = \min_{0 \leq r \leq m} \{L_{q-1}(i, r) + L(j_q, m - r)\}, 1 \leq q \leq k - 1,$$

$$L(i, m) = 0 \text{ if } L_{k-1}(i, m - 1) \leq \mu_i; \text{ and}$$

$$L(i, m) = L_{k-1}(i, m), \text{ otherwise.}$$

*Proof.* Fig. 2(c), 2(d) illustrate the basic concept of this theorem. To find an optimal server placement, we can view an intermediate node with  $k$  child nodes in Fig. 2(c) as the sub-tree in Fig. 2(d). Then, the minimum arrival rate of leakage requests,  $L(i, m)$ , can be computed recursively along the sub-tree. As the detailed proof of this theorem is similar to that of Theorem 3, it is omitted here.  $\square$

**Theorem 5.** The minimum number of I/O servers that meet their constraints can be obtained by finding the minimum  $m$  such that  $L(0, m) = 0$ .

Based on Theorems 2 to 4, we can compute the minimum arrival rate of leakage requests that start from leaf nodes and work toward the root node. After the minimum arrival rate of leakage requests that reach the root node has been computed, the minimum number of I/O servers that meet their constraints can be computed according to Theorem 5. The proposed algorithm is presented in Fig. 3.

**Algorithm** Minimum\_Leakage  
Input: 1. the arrival rate  $\lambda_i$  for all leaf nodes.  
2. the service rate  $\mu_i$  for all intermediate nodes.  
Output: the minimum arrival rate  $L(i, m)$  for  $0 \leq i, m \leq n$ .  
Procedure:  
1. sort all nodes according to their distance to the root node in decreasing order.  
2. for each node  $i$  do  
3. if node  $i$  is a leaf node then  
4. compute  $L(i, m) = \lambda_i$  for  $0 \leq m \leq n$   
5. else  
6. Let the child nodes of node  $i$  be nodes  $j_0, \dots, j_{k-1}$   
7. compute  $L_0(i, m) = L(j_0, m)$  for  $0 \leq m \leq n$   
8. for  $q$  from 1 to  $k - 1$  do  
9. compute  $L_q(i, m) = \min_{0 \leq r \leq m} \{L_{q-1}(i, r) + L(j_q, m - r)\}$  for  $0 \leq m \leq n$   
10. endfor  
11. for  $m$  from 0 to  $n$  do  
12. if  $L_{k-1}(i, m - 1) \leq \mu_i$  then  $L(i, m) = 0$  else  $L(i, m) = L_{k-1}(i, m)$  endif  
13. endfor  
14. endif  
15. endfor

**Fig. 3.** An optimal algorithm for the minimum server placement problem

In the first line of the algorithm, we sort all nodes according to their distances to the root node in decreasing order. This ensures that child nodes will be computed before their parents so that Theorems 2 to 4 can be correctly applied. The

execution time of this step is  $O(n \log n)$ . The loop in line 2 iterates over every node in the system. For each leaf node, it takes  $O(n)$  execution time in line 4. For an intermediate node that has  $k$  child nodes, it takes  $O(n^2)$  execution time in line 9, and iterates  $k - 1$  times in line 8. This results in  $O(kn^2)$  execution time for lines 8 to 10. Lines 11 to 13 also take  $O(n)$  execution time. Consequently, the complexity of lines 3 to 13 is  $O(kn^2)$ ; and the complexity of the whole algorithm is  $O(n^3)$ , where  $n$  is the number of nodes in the Grid system.

## 4 The Balanced Server Placement Problem

As mentioned in section 2 (the last paragraph), a major factor in the performance of a queuing system is the workload of the servers. Since each server may have a different capability, a server's workload is defined as the ratio of the arrival rate over the service rate. The minimum server problem sets a lower bound on the number of I/O servers. However, usually we would like to set up more I/O servers to reduce the workload. In this case, we are concerned with the maximum workloads of the I/O servers. In other words, we try to place a given number of servers appropriately so that the workload of the servers is as balanced as possible. We call this the *balanced server placement problem*.

**Definition 3.** The workload of a server  $i$ , denoted by  $\rho_i$ , is defined as the ratio of its arrival rate over its service rate:  $\rho_i = \lambda_i/\mu_i$

**Definition 4.** The maximum workload of a system is defined as the maximum workload among all servers in the system.

**Definition 5.** Given the network topology, request arrival rates and I/O service rates, the *balanced server placement problem* is: How to place a given number of I/O servers such that the maximum workload of the grid system is minimized?

Let  $m_0$  represent the lower bound on the number of I/O servers, assume there are  $m \geq m_0$  servers to be placed. Our goal is to place at most  $m$  servers such that the maximum workload is minimized. First, we present an algorithm to find a server placement when the maximum workload is known. Instead of solving this problem directly, we transform it into a minimum server placement problem discussed in section 3.

**Theorem 6.** There exists a placement of at most  $m$  servers such that  $\max\{\frac{\lambda_i}{\mu_i}\} \leq \rho$  if and only if the minimum number of servers needed for arrival rates  $\lambda_i$  and service rates  $\mu'_i = \rho \cdot \mu_i$  is less than or equal to  $m$ .

*Proof.* First, suppose that the minimum number of servers needed for arrival rates  $\lambda_i$  and service rates  $\mu'_i = \rho \cdot \mu_i$  is less than or equal to  $m$ . By Definition, there must exist a placement of at most  $m$  servers such that  $\lambda_i \leq \mu'_i = \rho \cdot \mu_i$  for all server nodes  $i$ . Thus,  $\lambda_i/\mu_i \leq \rho$  for all server nodes  $i$ . Accordingly, we can derive  $\max\{\lambda_i/\mu_i\} \leq \rho$ . This completes the proof of the "if" part.

Next, suppose there exists a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho$ . We can derive  $\lambda_i/\mu_i \leq \rho_i$  and  $\lambda_i \leq \rho \cdot \mu_i$  for all server nodes  $i$ . Therefore, the minimum number of servers needed for arrival rates  $\lambda_i$  and service rates  $\mu'_i = \rho \cdot \mu_i$  must be less than or equal to  $m$ . This completes the proof of the only if part.  $\square$

**Theorem 7.** If there is no placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho$  and  $\rho' \leq \rho$ , then there cannot be a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho'$ .

*Proof.* We prove this theorem by contradiction. Assume that there is no placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho$  and  $\rho' \leq \rho$ , there exists a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho'$ . Accordingly, we can derive that there must exist a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho' \leq \rho$ . However, this contradicts the assumption. Therefore, there cannot be a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho'$ .  $\square$

According to Theorem 6, we can determine if there exists a placement of at most  $m$  servers such that  $\max\{\lambda_i/\mu_i\} \leq \rho$  by using the algorithm for the minimum server placement problem in section 3. The main difficulty with this approach is that we do not know the optimal value of the maximum workload yet. Fortunately, Theorem 7 provides a foundation for searching the optimal value of the maximum workload. It implies that if there is no server placement for a maximum workload  $\rho$ , then the optimal value must be greater than  $\rho$ . On the other hand, if there is a server placement for a maximum workload  $\rho$ , then it is possible to further minimize the value of the maximum workload. Combining Theorem 6 and 7 allows us to find the optimal value through a binary search of the maximum workload.

Before applying a binary search, however, we have to determine an upper bound and a lower bound. It is rather easy to get an upper bound and a lower bound on the maximum workload. So long as  $m \geq m_0$ , there always exists an upper bound of 1 on the maximum workload. A lower bound can be computed by assuming that the fastest  $m$  servers are chosen and I/O requests are distributed to these servers evenly. Next, we can combine a binary search of the maximum workload and the algorithm for the minimum server placement problem to find the optimal value of the maximum workload. Because the upper bound of the binary search is a constant and the lower bound is a function of the input parameters, the workload-balance algorithm is strongly polynomial.

Our algorithm can be further generalized to consider network bandwidth. Take Fig. 2(a) and 2(b) as examples. The arrival rate of I/O requests that pass through the communication channel between node  $j$  and node  $i$  is denoted as  $L(j, p)$ . Let  $\mu_{ji}$  be the service rate of this communication channel. To meet the constraint of the communication channel, it is desirable that  $L(j, p) \leq \mu_{ji}$  in the minimum server placement problem and  $L(j, p) \leq \rho \cdot \mu_{ji}$  in the balanced server placement problem.

## 5 Experimental Results

To evaluate the performance of the proposed algorithms, we conducted several experiments in which 1000 test cases based on the proposed Grid model are randomly generated. The number of nodes in each case is approximately 1000. The arrival rates for the leaf nodes and the service rates for intermediate nodes are generated from a negative exponential distribution. We also implemented a heuristic-based greedy method similar to that proposed in [12] as a reference.

The experimental results for the minimum server placement problem, shown in Fig. 4, compare the performance of the proposed optimal algorithm with the heuristic-based greedy method. The performance metric is the difference in the number of servers by the proposed optimal algorithm and the greedy method. The vertical axis shows the number of test cases, while the horizontal axis shows the difference in the number of servers used by the two methods.

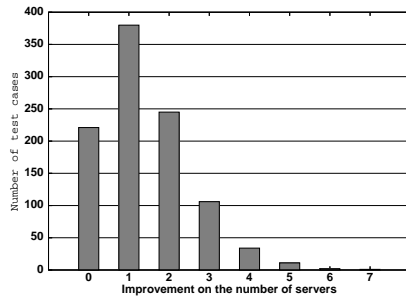
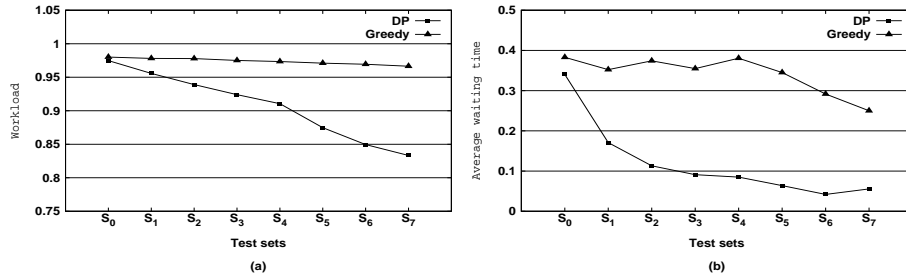


Fig. 4. Performance comparison for the minimum server problem

According to the experimental results, the greedy method can only generate an optimal solution for 22.1% of the test cases. The optimal solution generated by our algorithm uses one less server than the greedy method in 38% of the test cases and two or less servers than the greedy method in 39.9% of test cases. Based on the results in Fig. 4, we classified the 1000 test cases into seven sets for use in the following experiments. Thus, test set  $S_i$  contains those test cases in which our algorithm uses  $i$  less servers than the greedy method.

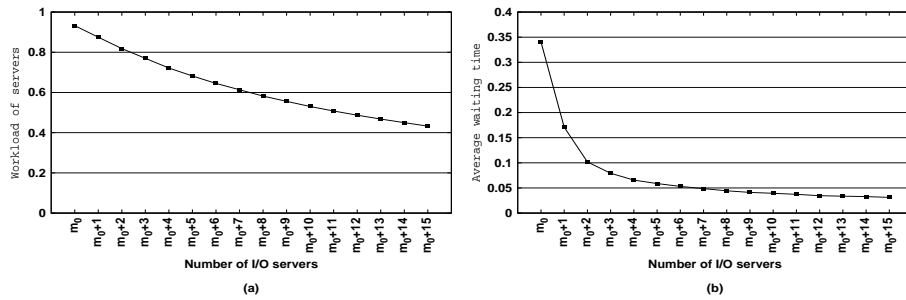
Fig. 5(a) shows the workloads of the greedy method and the optimal algorithm. For each test set, the optimal algorithm and the greedy method use the same number of servers, and we take the average of maximum workloads of the optimal algorithm and the greedy method as the performance metric in this experiment. It is obvious that the the difference of the workloads becomes larger when the difference of the minimum numbers of servers required by the two algorithms increases. This means that, when using the same number of I/O servers, the optimal algorithm can actually reduce the maximum workload of the I/O servers and therefore balance their workload better than the greedy method.



**Fig. 5.** (a) The workloads of the optimal algorithm and the greedy method (b) The average waiting times of different test sets

Next, we compare the average waiting times of the two algorithms. The results are shown in the Fig. 5(b). This experiment demonstrates the major benefit of our optimal algorithm. The results show that, using the same number of servers as the greedy method, our algorithm reduces the average waiting time of the grid system dramatically compared to the greedy method.

Fig. 6(a) shows the maximum workload of the optimal algorithm as the number of I/O servers increases, where  $m_0$  is the lower bound on the number of I/O servers for test cases in  $S_0$ . It is clear that the maximum workload decreases as the number of I/O servers increases. This data can help us determine an appropriate number of servers in a grid system. Fig. 6(b) also shows the average waiting time of the optimal algorithm as the number of I/O servers increases. It can also help us to determine an appropriate number of servers in a grid system when the average waiting time is the major concern.



**Fig. 6.** The workload and the average waiting time versus the number of I/O servers

## 6 Conclusions

In this paper, we focus on some server placement problems in Data Grid environments. Given a hierarchical network with requests from clients and constraints on server capability, the minimum server placement problem attempts to place the minimum number of servers that can deal with clients requests. As our model allows servers have different I/O capabilities, it is more general than similar work in the literatures. Instead of using a heuristic approach, we propose an optimal algorithm based on dynamic programming as a solution to this problem.

Next, we consider the balanced server placement problem, which tries to place a given number of servers appropriately so that the workload of the servers is as balanced as possible. We show that optimal server placement can be achieved by combining the above algorithm with a binary search of workloads. Finally, we extend the above approach so that constraints on network capability can also be dealt with. The experiment results clearly show an improvement on the number of servers and the maximum workload. As the maximum workload is reduced, the waiting time is also reduced.

## References

1. Foster, I.T., Kesselman, C., Tuecke, S.: The Anatomy of the Grid : Enabling Scalable Virtual Organizations. *The International J. of High Performance Computing* **15**(3) (2001)
2. Johnston, W.E.: Computational and data Grids in large-scale science and engineering. *Future Generation Computer Systems*. **18**(8) (2002) 1085–1100
3. EU DataGrid. (<http://www.edg.org>)
4. PPDG: Particle Physics Data Grid. (<http://www.ppdg.net>)
5. iVDGL: International Virtual Data Grid Laboratory. (<http://www.ivdgl.org>)
6. Deelman, E., Kesselman, C., Mehta, G., Meshkat, L., Pearlman, L., Blackburn, K., Ehrens, P., Lazzarini, A., Williams, R., Koranda, S.: GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists. In: *HPDC 2002*. (2002)
7. BIRN: The Biomedical Informatics Research Network. (<http://www.nbirn.net>)
8. Hoschek, W., Jaén-Martínez, F.J., Samar, A., Stockinger, H., Stockinger, K.: Data Management in an International Data Grid Project. In: *GRID 2000*. (2000) 77–90
9. Ranganathan, K., Foster, I.T.: Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In: *GRID 2001*. (2001) 75–86
10. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications* **23**(3) (2000) 187–200
11. Lamahmedi, H., Shentu, Z., Szymanski, B.K., Deelman, E.: Simulation of Dynamic Data Replication Strategies in Data Grids. In: *IPDPS 2003*. (2003) 100
12. Abawajy, J.H.: Placement of File Replicas in Data Grid Environments. In: *International Conference on Computational Science*. (2004) 66–73
13. Bell, W.H., Cameron, D.G., Carvajal-Schiaffino, R., Millar, A.P., Stockinger, K., Zini, F.: Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In: *International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003*. (2003) 120–126
14. Grid Physics Network (GriPhyN). (<http://www.griphyn.org>)