J Supercomput (2007) 42: 267–282
DOI 10.1007/s11227-007-0118-4

Optimizing server placement in hierarchical grid environments

Chien-Min Wang · Chun-Chen Hsu · Pangfeng Liu · Hsi-Min Chen · Jan-Jan Wu

Published online: 19 April 2007

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 requests from clients. 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 on workloads. This approach can be further extended to deal with constrains on network capability. The simulation results clearly show the improvement in the number of servers and the maximum workload. Furthermore, as the maximum workload is reduced, the waiting time is reduced accordingly.

C.-M. Wang (✉) · J.-J. Wu
Institute of Information Science, Academia Sinica, Nankang 115, Taipei, Taiwan
e-mail: cmwang@iis.sinica.edu.tw

J.-J. Wu
e-mail: wuj@iis.sinica.edu.tw

C.-C. Hsu · P. Liu
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

C.-C. Hsu
e-mail: d95006@csie.ntu.edu.tw

P. Liu
e-mail: pangfeng@csie.ntu.edu.tw

H.-M. Chen
Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan
e-mail: seeme@selab.csie.ncu.edu.tw

Keywords File replication · Grids · Queueing systems · Server placement · Waiting time · Workload

1 Introduction

Grid technologies, which enable scientific applications to utilize a wide variety of distributed computing and data resources, are classified into two categories: Computing Grids and Data Grids [9, 13]. 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 [8], PPDG [15], iVDGL [12], GriPhyN [6] and BIRN [3].

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 [4, 5, 7, 11, 14, 16, 17]. 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 [14], response time and bandwidth consumption [16].

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 the minimum server placement problem attempts to place the minimum number of servers that can satisfy requests from clients. 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. This approach can be further extended to deal with constrains on network capability. The experimental results clearly show the improvement in the number of servers and the maximum workload. Furthermore, as the maximum workload is reduced, the waiting time is also reduced.

2 Background

In this paper, we use a hierarchical Grid model, one of the most common architectures in current use [1, 2, 10, 11, 16]. 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 the 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 [2, 16].

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

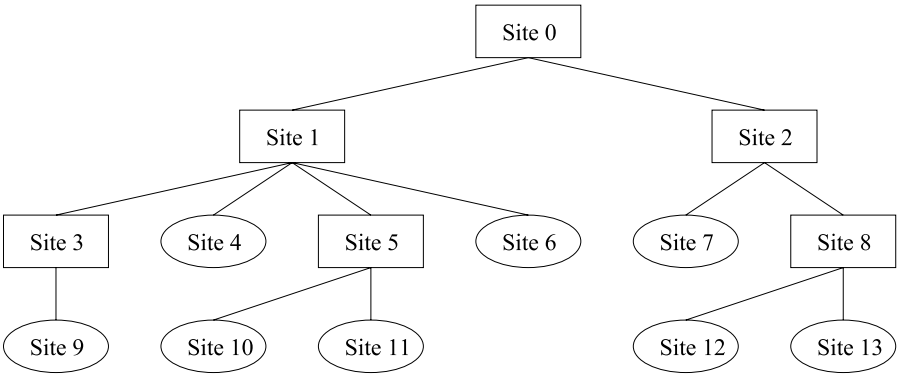

Fig. 1 The hierarchical Grid model

and passes through routers until it reaches 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 a 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 λ_j that represents the arrival rate of I/O requests. λ_j can be computed 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.

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 workloads of I/O servers are 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 requests 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 among servers and balance their workloads. However,

it is quite expensive to setup I/O servers in a Grid system, as having more servers usually lead to lower utilization, which means a waste of the system's resources and increased maintenance costs. Therefore, our first problem is to place the minimum number of I/O servers that will satisfy I/O requests from clients.

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, it is intuitively to employ a greedy method, similar to that in [1], 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 $L(i, m) = \lambda_i$ 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 node's parent. By definition, $L(i, m) = \lambda_i$ for $0 \leq m \leq n$. □

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 happens if and only if $\min_{0\le r\le m-1}\{L(j,r)+L(k,m-r-1)\}\le\mu_i$. 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:

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

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 $0\le p,q\le m-1$ and $p+q\le m-1$. 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 $L(j,p)+L(k,q)\le\mu_i$. According to Theorem 1, we can derive: $\mu_i\ge L(j,p)+L(k,q)\ge L(j,p)+L(k,m-1-p)\ge\min_{0\le r\le m-1}\{L(j,r)+L(k,m-r-1)\}$. 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. 2b. Obviously, we have the inequalities $0\le p,q\le m$ and $p+q\le m$. 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: $L(i,m)=L(j,p)+L(k,q)\ge L(j,p)+L(k,m-p)\ge\min_{0\le r\le m}\{L(j,r)+L(k,m-r)\}$. According to above the assumption, this is an optimal server placement. Hence, all the equalities must hold. This completes the proof of case 2. □

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

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

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

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

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

Proof Figures 2c and 2d 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. 2c as the sub-tree in Fig. 2d. Then, the minimum arrival rate of leakage requests, $L(i,m)$,

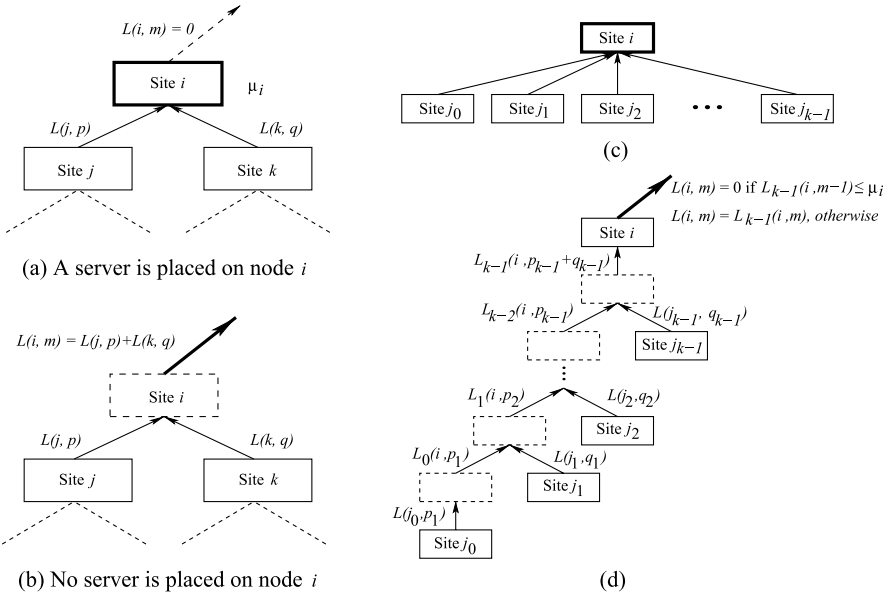

Fig. 2 **a** and **b** illustrate two possible server placements on node i. **c** and **d** illustrate the basic concept of Theorem 4

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. □

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 by starting from leaf nodes and working 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.

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.

Algorithm Minimum_Leakage
Input: 1. the arrival rate λ_i for all leaf nodes.
2. the service rate μ_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 $j_0, \ldots, j_{k-1}$
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

4 The balanced server placement problem

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

Definition 3 The workload of a server i, denoted by ρ_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 aims at placing a given number of I/O servers so 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 Sect. 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* λ_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 λ_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 $\lambda_i/\mu_i \leq \rho$ and $\lambda_i \leq \rho \cdot \mu_i$ for all server nodes i. Therefore, the minimum number of servers needed for arrival rates λ_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". □

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 $\rho' \leq \rho$. If there exists a placement of at most m servers such that $\max\{\lambda_i/\mu_i\} \leq \rho'$, 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 $\max\{\lambda_i/\mu_i\} \leq \rho'$. □

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 Sect. 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 ρ, then the optimal value must be greater than ρ. On the other hand, if there is a server placement for a maximum workload ρ, then it is possible to further minimize the value of the maximum workload. Combining Theorems 6 and 7 allows us to find the optimal value through a binary search on 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. 4 as an example. Let μ_{ji} be the service rate of the communication channel

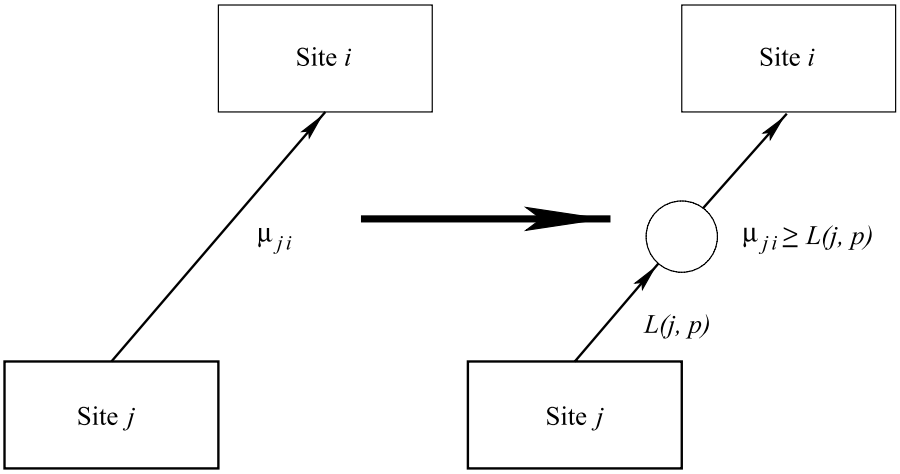

Fig. 4 An example of network bandwidth

between node j and node i. The arrival rate of I/O requests that pass through the communication channel between node j and node i is denoted as $L(j, p)$. 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

In this section we conduct several experiments to evaluate the performance of the proposed algorithms. Test cases are generated based on the proposed Grid model. The height of each case is at most 8. Each node has at most 4 children. 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 uniform distribution. There are four groups of test cases. Each group has a different range of service rate: 0–400, 50–350, 100–300 and 150–250. We will refer those groups as group 1, 2, 3 and 4, respectively. There are 1000 test cases in each group.

We compare the performance of the three algorithms: Greedy [1], Workload_Based and Waiting_Time_Based algorithms. The Workload_Based (WB) algorithm is described in Sect. 4. The Waiting_Time_Based (WTB) algorithm is similar to WB except it tries to minimize the maximum waiting time of M/M/1 queueing systems. In other words, it tries to minimize $\max_i\{1/(\mu_i - \lambda_i)\}$. Table 1 shows the experimental parameters.

The experimental results for the minimum server placement problem is shown in Fig. 5. Here we only compare Greedy and WB because WTB uses the same number of servers as WB. The performance metric is the difference in the number of servers used by WB and Greedy, i.e., the extra number of servers used by the Greedy algorithm. The vertical axis shows the number of test cases, while the horizontal axis shows the difference in the number of servers used by these two algorithms.

Table 1 Parameters of experiments

Parameter	Description
Height of tree	≤ 8
Number of child nodes	≤ 4
Number of nodes in each case	≈ 1000
Range of service rates	0–400, 50–350, 100–300 and 150–250
Algorithms	Greedy, WB and WTB

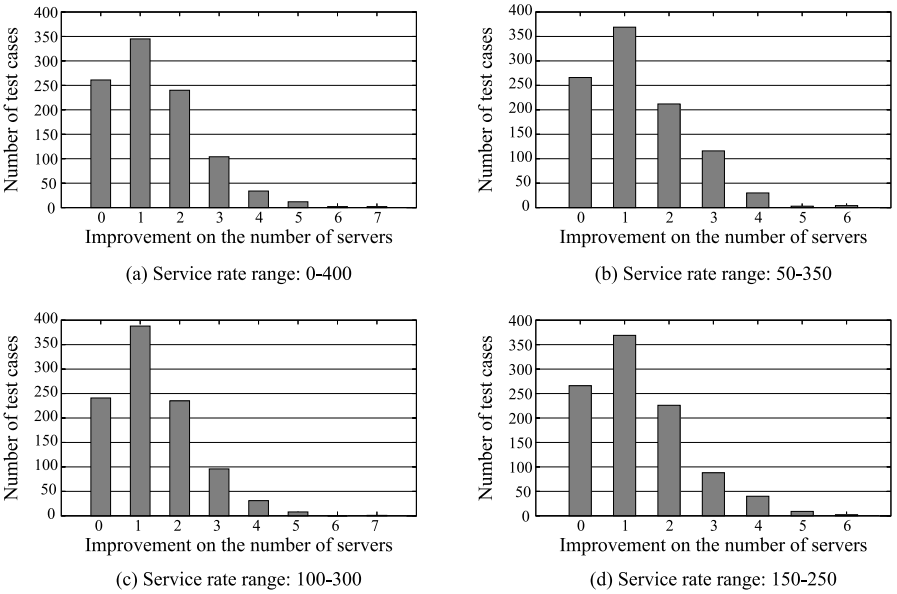

Fig. 5 Performance comparison for the minimum server problem

Figure 5a shows that Greedy can only generate an optimal solution for 26.1% of the test cases when service rates are between 0 and 400. WB uses one less server than Greedy in 34.5% of the test cases and two or less servers than the greedy method in 39.4% of test cases. Figures 5b, 5c and 5d also show similar results. Based on the results in Fig. 5, we further classify the 1000 test cases in each group into seven additional sets for use in the following experiments. Thus, test set S_i contains those test cases in which our algorithms use i less servers than the Greedy algorithm.

Figure 6 shows the workloads of the three algorithms. For each test set, WB and WTB use the same number of servers as Greedy. Figure 6 uses the average of maximum workloads of each test set as the performace metric. It is obvious that the difference between Greedy and our algorithms becomes more significant as the difference in the minimum numbers of servers increases. This means that, when using the same number of I/O servers, both WB and WTB can actually reduce the maximum workload of the I/O servers and therefore balance their workloads better than the Greedy algorithm.

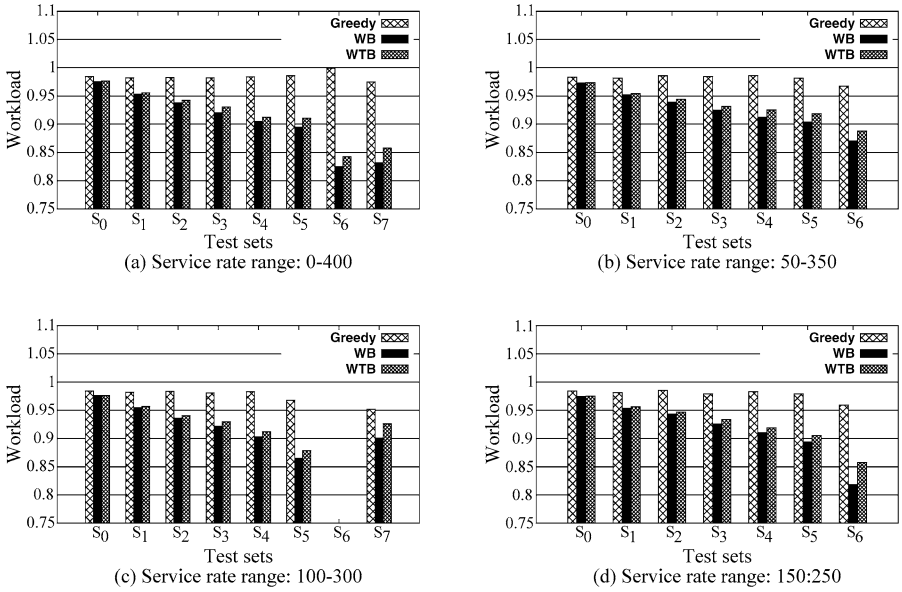

Fig. 6 The average of maximum workloads

Other groups show similar results. This indicates that our algorithms also perform better than Greedy in other ranges of service rates. The workloads of WB are slightly better than those of WTB. This is because WB is designed to minimize the maximum workload while WTB is to minimize the maximum waiting time. The difference, however, is not significant. Note that in Fig. 6c there is no data in set S_6. It is because no test case in group 3 uses 6 more servers in the Greedy algorithm.

Before we compare the average waiting times of algorithms. we should take some points into account. In our experiments, we find that there are extraordinarily long waiting times in several test cases. This happens when the service rate of a server is very close to its request rate in a heavy loaded system. We refer those cases as outliers. For example, the maximum waiting time of an outlier in group 3 reaches 90.9 while the average of maximum waiting times is only 0.968. To prevent those outliers producing confusing results, we introduce an additional parameter, ϵ, to limit the maximum waiting time of test cases. In the following experiments, it is desired that $\lambda_i \leq \mu_i - \epsilon$ for each server i.

In the presence of ϵ, we can still use the binary search technique to find the maximum workload described in Sect. 4 with minor modifications. Suppose we want to know if $\max_i\{\lambda_i/\mu_i\} \leq \alpha$ and $\lambda_i \leq \mu_i - \epsilon$, we can let $\mu_i' = \min(\alpha\mu_i, \mu_i - \epsilon)$. Similarly, suppose we want to know if $\max\{1/(\mu_i - \lambda_i)\} \leq \beta$ and $\lambda_i \leq \mu_i - \epsilon$, we can let $\mu_i' = \mu_i - \max\{1/\beta, \epsilon\}$. Then, we can use the dynamic programming algorithm described in Sect. 3 to deal with that problem.

We now discuss the influence of ϵ. Intuitively, the larger ϵ is, the more servers are needed. Figure 7 shows the extra number of servers needed for different ϵ. When $\epsilon = 0.05$, there are 997 test cases using the same number of servers as $\epsilon = 0$. When $\epsilon = 1$, only 831 test cases use the same number of servers as $\epsilon = 0$. Accord-

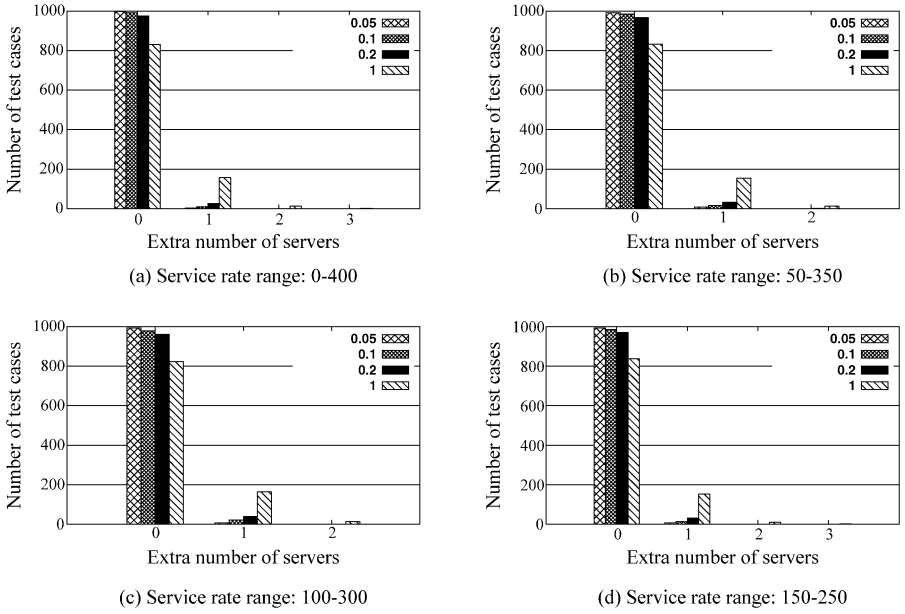

Fig. 7 The extra number of servers with different ϵ values

ing to these results, a smaller ϵ is preferred since we tend to use a less number of servers.

The functionality of ϵ is to filter out those test cases with extraordinarily long waiting time. Therefore we would like to see the effect of ϵ. Figure 8 shows the maximum of maximum waiting times of WTB in each group with different ϵ. From Fig. 8 we can see clearly the effect of ϵ. For example, in group 3, the maximum of the waiting time is 90.9 when $\epsilon = 0$ and 18.18 when ϵ is only 0.05. We choose $\epsilon = 0.05$ in our experiments since it can filter out those most outliers and in 99% cases it uses the same number of servers as $\epsilon = 0$.

After introducing ϵ, we then look at the results in the Fig. 9. The performace metric is the average of maximum waiting time of each test set. This result demonstrates the major benefit of our algorithms. The results show that, using the same number of servers as Greedy, our algorithms reduce the average waiting time of the grid system dramatically. For example, in Fig. 9a the average maximum waiting time of WTB is 0.17 while the average maximum waiting time of Greedy is 1.19 in set S_1. The WTB performs slightly better than WB in this metric. Again, all the test groups show the similar results.

Figure 10 shows the maximum workloads and the maximum waiting times of WB and WTB algorithms 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. These data can help us determine an appropriate number of servers in a grid system. 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.

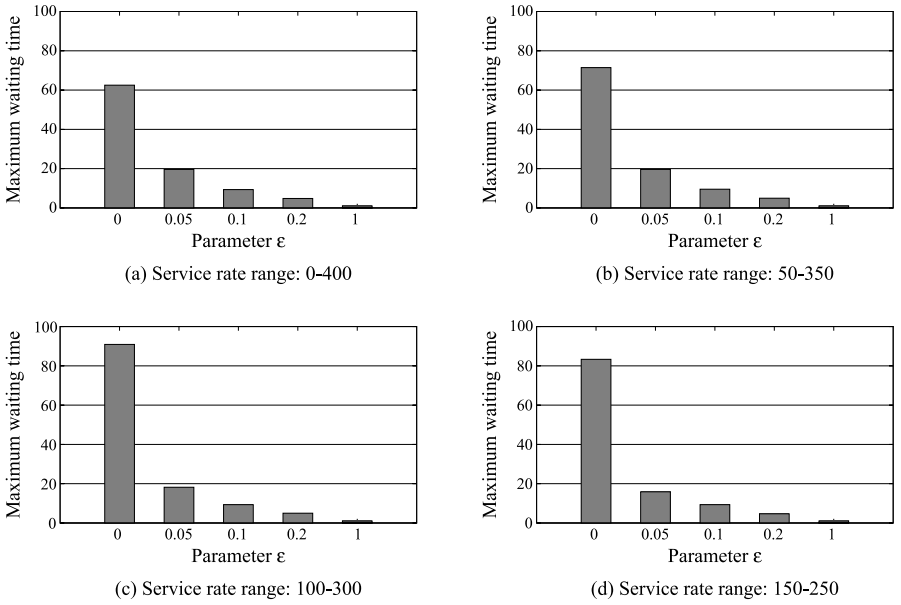

Fig. 8 The effect of ϵ on the maximum waiting time

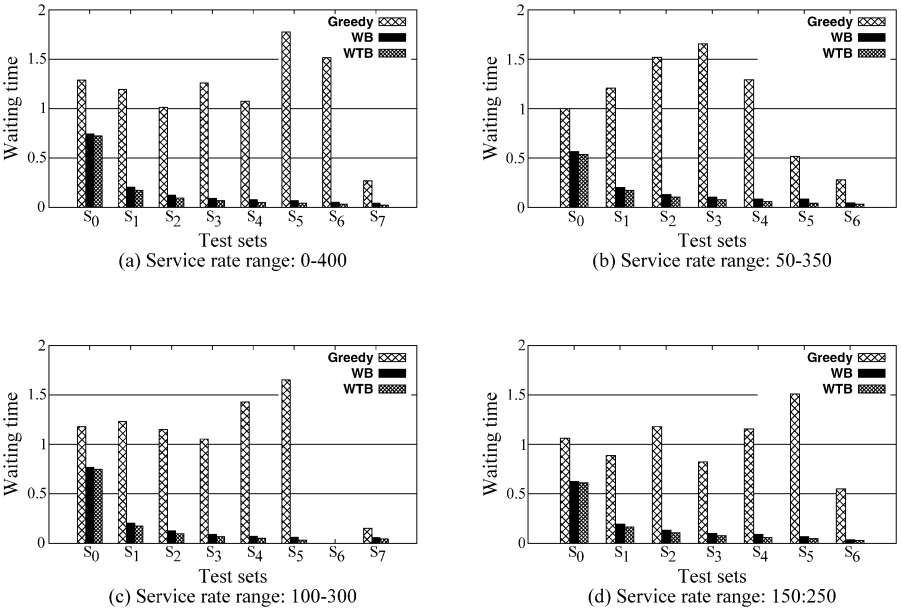

Fig. 9 The average of maximum waiting times

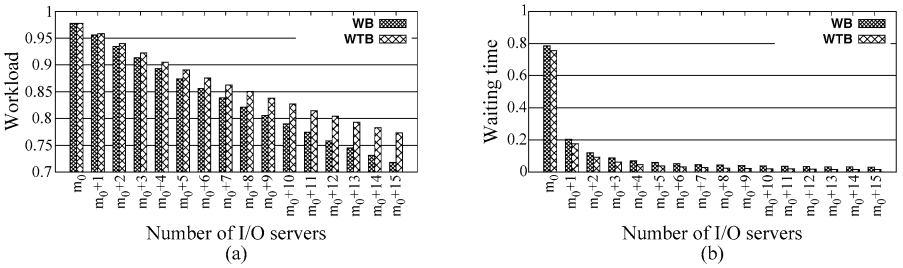

Fig. 10 The workload and the average waiting time versus the number of I/O servers. Server capacity range: 0–400

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 requests from clients. 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.

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

Acknowledgements The authors would like to thank the anonymous referees for their helpful suggestions. The authors also acknowledge the National Center for High-performance Computing in providing resources under the national project, "Taiwan Knowledge Innovation National Grid." This research is supported in part by the National Science Council, Taiwan, under Grant NSC 94-2213-E-001-023.

References

1. Abawajy JH (2004) Placement of file replicas in data grid environments. In: International conference on computational science, 2004, pp 66–73
2. Bell WH, Cameron DG, Carvajal-Schiaffino R, Millar AP, Stockinger K, Zini F (2003) 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, May 2003, pp 120–126
3. BIRN: the biomedical informatics research network. http://www.nbirn.net
4. Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S (2000) The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J Netw Comput Appl 23(3):187–200
5. Chervenak A, Schuler R, Kesselman C, Koranda S, Moe B (2005) Wide area data replication for scientific collaborations. In: SC'05: Proc the 6th IEEE/ACM international workshop on grid computing CD, Seattle, Washington, USA, IEEE/ACM, Nov 2005, pp 1–8

6. Deelman E, Kesselman C, Mehta G, Meshkat L, Pearlman L, Blackburn K, Ehrens P, Lazzarini A, Williams R, Koranda S (2002) GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In: HPDC 2002
7. Deris MM, Abawajy JH, Suzuri HM (2004) An efficient replicated data access approach for large-scale distributed systems. In: CCGRID, 2004, pp 588–594
8. EU DataGrid. http://www.edg.org
9. Foster IT, Kesselman C, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. Int J High Perform Comput 15(3)
10. Grid Physics Network (GriphyN). http://www.griphyn.org
11. Hoschek W, Jaén-Martínez FJ, Samar A, Stockinger H, Stockinger K (2000) Data management in an international data grid project. In: GRID 2000, pp 77–90
12. iVDGL: international virtual data grid laboratory. http://www.ivdgl.org
13. Johnston WE (2002) Computational and data grids in large-scale science and engineering. Future Gener Comput Syst 18(8):1085–1100
14. Lamehamedi H, Shentu Z, Szymanski BK, Deelman E (2003) Simulation of dynamic data replication strategies in data grids. In: IPDPS 2003, p 100
15. PPDG: particle physics data grid. http://www.ppdg.net
16. Ranganathan K, Foster IT (2001) Identifying dynamic replication strategies for a high-performance data grid. In: GRID 2001, pp 75–86
17. Ranganathan K, Iamnitchi A, Foster IT (2002) Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In: CCGRID, 2002, pp 376–381

Chien-Min Wang received a B.S. and a Ph.D. in Electrical Engineering from National Taiwan University in 1987 and 1991, respectively. Since then he joined the Institute of Information Science as an assistant research fellow. In January 1996, he became an associate research fellow of the institute. His major research interest includes parallel compilers, parallel algorithms, parallel computer architectures, and object-oriented technology. Dr. Wang is a member of IEEE Computer Society.

Chun-Chen Hsu received the B.S. in Psychology in 2002 and M.S. in Computer Science and Information Engineering in 2004, both from Taiwan University. He is now a Ph.D. student in the Department of Computer Science and Information Engineering of National Taiwan University.

Pangfeng Liu received the B.S. degree in Computer Science from National Taiwan University in 1985, and the M.S. and Ph.D. degrees in Computer Science from Yale University in 1990 and 1994, respectively. He is now a professor in the Department of Computer Science and Information Engineering of National Taiwan University. His research interests include parallel and distributed computing, randomized algorithms, and object-oriented methodology. He is a member of the ACM and the IEEE.

Hsi-Min Chen received the M.S. degree in Computer Science and Information Engineering from National Central University, Taoyuan, in 2000. He has been a software engineer at Institute of Information Science, Academia Sinica, Taiwan, since 2001. He is also a Ph.D. student in the Department of Computer Science and Information Engineering at National Central University. His major research interest includes software engineering, computer supported cooperative work, service-oriented software technology, and grid computing.

Jan-Jan Wu received the B.S. degree and the M.S. degree in Computer Science from National Taiwan University in 1985 and 1987. In 1995, she received the Ph.D. in Computer Science from Yale University. From August 1994 to October 1995, she worked as a consultant in Cooperating Systems Corporation, participating in compiler optimizations for High Performance Fortran. In November 1995, she joined the Institute of Information Science as an assistant research fellow, and has been an associate research fellow since Oct. 2000. Dr. Wu's research interests include compilers and runtime support for parallel and distributed computation, cluster computing, and parallel data structures. She is a member of ACM and IEEE.