CHAPTER IV

Experiments and Results

In this chapter, a set of experiments to evaluate the proposed algorithm is described. There are four sections in this chapter. The first three sections are experiments to compare the performance of the proposed algorithms. The other is the study of the effect of variance of transmission rate on the proposed algorithms. These experiments are performed on a simulated grid.

A grid simulation is implemented by using a C-based simulation language called PARSEC [27] (Parallel Simulation Environment for Complex Systems) to evaluate the performance of the proposed algorithms. Parallel data transfer and partial file transfer, which are features of GridFTP [28], are also allowed in this simulation.

The grid systems simulated for the experiments in this chapter contains one client which requests for a dataset, one co-allocator, and five servers. The client, the co-allocator and five servers are connected via links whose available bandwidths are random variables with normal distribution. In one environment, links between sites in the grid have different average bandwidths. In the other environment, all links between sites have equal average bandwidths. The average available bandwidth in both environments is shown in Table 1. Distances between all sites are 100 km. Two grid environments are simulated. Transmission delay, data processing delay and distances between the client and servers are considered in the simulation. However, the processing power and available bandwidth are unknown to the co-allocator.

The dataset used in the experiments contains five fragments. The fragment sizes are different for each experiment, and are specified in each section. Fragments of the dataset are replicated randomly on the servers.

Table 1: The average bandwidths for two grid environments

KB/sec	S_1	S_2	S_3	S_4	S_5
The first environment	375	500	500	175	175
The second environment	375	375	375	375	375

For each experiment, 30 requests are simulated to measure the completion time and the average waiting time. For each request, the fragment placement is varied so that each co-allocation strategy is not penalized by a placement which decreases its performance.

The completion time is measured from starting the first block transmission to the server completes the last block transmission. The average waiting time is the average time each server spends on waiting for the last server to finish transmission. From Figure 13, the completion time is t_5, and the average waiting time is $\frac{\sum_{i=1}^{n} w_i}{n} = \frac{w_1 + w_2 + w_4 + w_5}{5}$ where w_i is the waiting time at the server S_i.

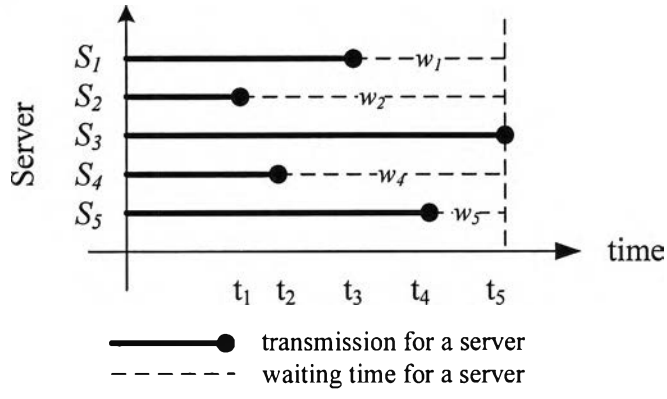


Figure 13: The calculation of completion time and waiting time

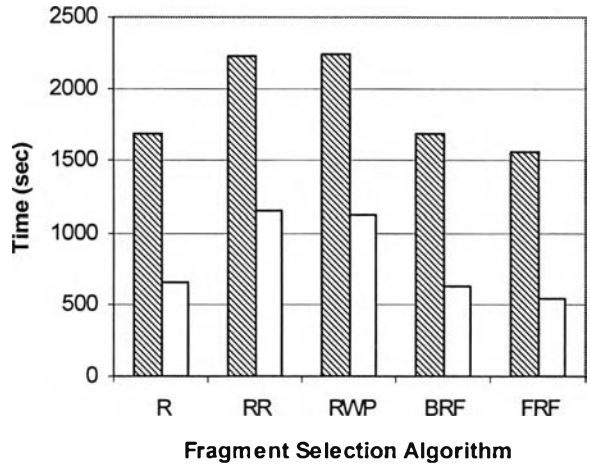
Distances between the client and servers are used to calculate router delay time. One router causes 1 millisecond (mS) delay for a transmission. The routers are placed on the way between the client and servers every 0.5 kilometers. Data processing delay of 1 millisecond (mS) is taken into account for a block transmission.

4.1 Performance for Fragments with Different Sizes and Different Numbers of Replicas

The experiments in this section aim to compare the performance of the proposed strategies in a normal situation, i.e., fragments of data are of various sizes, and different numbers of replicas are created for different fragment. The sizes of five fragments are 100, 200, 300, 400, and 500 MB. The number of replicas of each fragment is random, and each replica is randomly placed on the five servers. For available bandwidths of links in both grid environments, the coefficient of variation of is 50%.

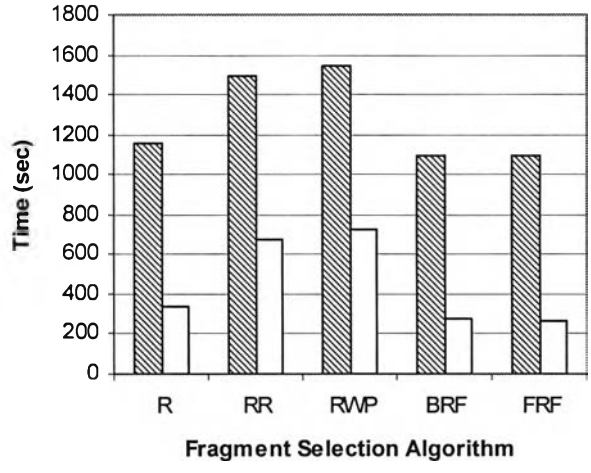
From both grid environments, the experiment shows that Fewest-replicas-first algorithm yields the best average completion time as shown in Figure 14 and Figure 15. Comparing to Random algorithm, Fewest-replicas-first algorithm yields approximately 6% better in both environments. Comparing to Biggest-remaining-first algorithm, Fewest-replicas-first algorithm yields better approximately 7% in the first environment and it is not significantly better in the second environment. The Round-robin and Random-with-weighted-probability algorithms are the worst.

From the experiment, it shows that Round-robin algorithm and Random-with-weighted-probability algorithms perform worse than the other three. Random-with-weighted-probability algorithm chooses a fragment based on the original fragment size, which does not reflect the current workload of each server. This results in the worst performance among all five algorithms. Round-robin algorithm selects each replica, among all available locally, in turns. As a result, a fragment with more replication is chosen more often. If a server has small fragments with more replication, it finishes the transmission earlier and is idle while other severs continue the transmission.



R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

Figure 14: Average completion time and waiting time in grid that has different bandwidths



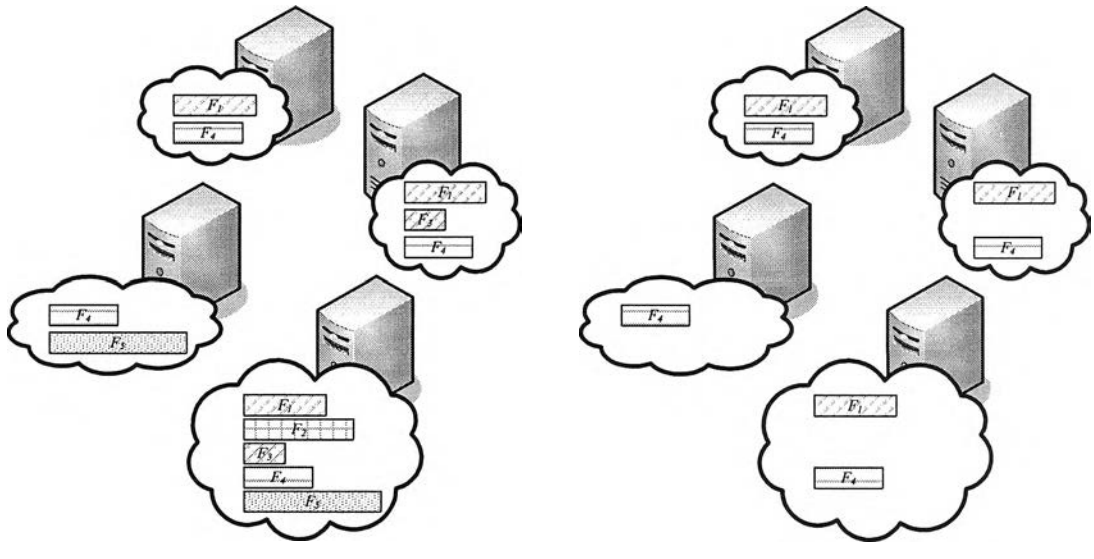
R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

Figure 15: Average completion time and waiting time in grid that has same bandwidth

The following is a reason why Fewest-replicas-first algorithm yields approximately 6% better than Random algorithm. Fewest-replicas-first algorithm sends blocks from a fragment that has fewest replicas first. At the beginning of the transmission, fragments which have fewer replicas are sent. Later, when a server

finishes, it sends fragments with more replication. This means that there are more servers to transfer the fragments with more replication. Many servers can send data in parallel and the waiting time can be decreased.

For example, there are five fragments of data distributed on grid containing four servers as shown in Figure 5. At the beginning of the transmission, all four servers have fragments of data which are needed to be transferred to the client as shown in Figure 16 (a), and all four servers can transfer the data in parallel. Since Fewest-replicas-first algorithm sends the block from the fragments that have fewest replicas first, the fragments with fewer replicas tend to be completely transferred to the client first. Thus, the fragments with more replicas are left to be transferred in the later phrase. For example, in Figure 16 (b) fragments F_2, F_3, and F_5, which have one and two replicas in the grid, are completely transferred, and fragments F_1 and F_4, which are replicated on four servers, are left in the last phrase. That is four servers transfer fragments F_1 and F_4 in the last phrase of the transmission.

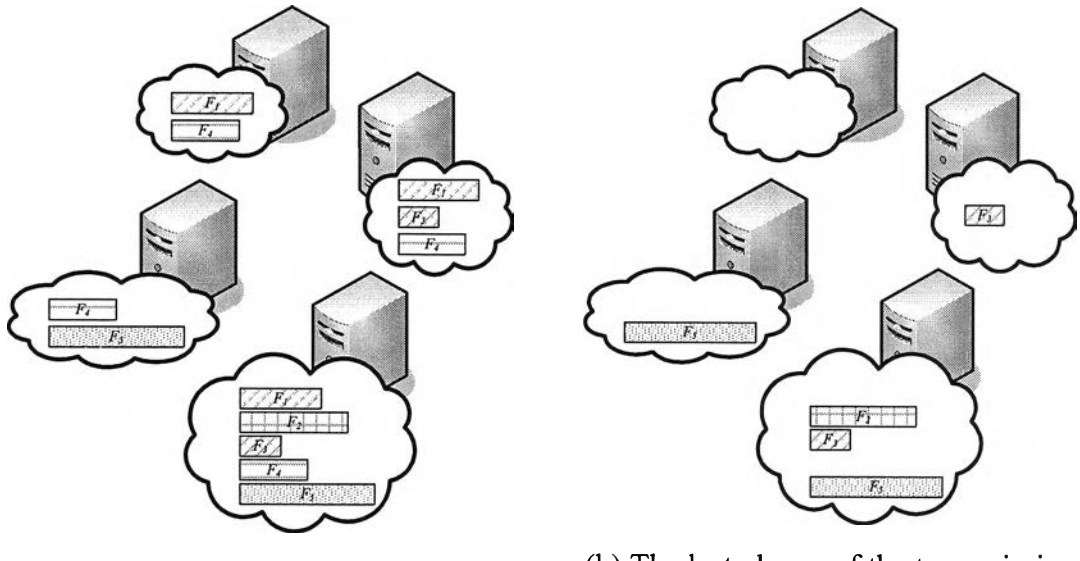


(a) The beginning of the transmission (b) The last phrase of the transmission

Figure 16: The beginning and last phrase of the transmission when using Fewest-replicas-first algorithm

On the contrary, if the servers send blocks from the fragments that have more replicas first, the transmission of these fragments is finished in the earlier phase. Later, there are fewer servers left to transfer fragments that have fewer replicas. As a

result, servers that do not have these replicas are idle and wait for only few servers to complete the transmission. This causes higher completion time and waiting time for other servers.



(a) The beginning of the transmission (b) The last phrase of the transmission

Figure 17: The beginning and last phrase of the transmission when fragments with more replication are sent first

In the similar situation as in the previous example, at the beginning of the transmission, all four servers have fragments of data which are needed to be transferred to the client as shown in Figure 17 (a), and all four servers can transfer the data in parallel. If the fragments with more replication are sent first, the fragments with fewer replications left in the later phrase. For example, fragments F_1 and F_4 with three and four replications are completely transferred earlier. Fragments F_2, F_3, and F_5 are left in the last phrase with three servers to be transferred. Only three servers transfer fragments F_2, F_3, and F_5 in the last phrase of the transmission as shown in Figure 17 (b). One server is idle because fragments located in the server are completely transferred. This causes the higher completion time comparing to Fewest-replicas-first algorithm because of fewer servers in the last phrase.

Biggest-remaining-first algorithm tries to distribute the workload by sending a fragment with bigger portion of untransmitted data first in order to make all fragments have approximately the same amount of untransmitted data. It tries to make all

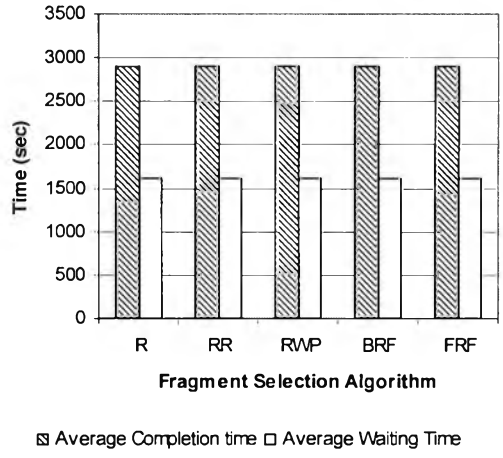
fragments finished transferring at the same time. If fragments are distributed evenly among all servers, no server spends idle time waiting finish while others have more data to be transmitted.

From the experiments, Biggest-remaining-first algorithm performs well in the grid with the same bandwidth. The algorithm tries to equalize the amount of unsent data in each fragment. Thus, the amounts of unsent data in all fragments are approximately the same in the later phrase, and most servers are not idle at the end. Because all links in the grid have the same bandwidth, all servers spend roughly the same amount of time to transfer an assigned block.

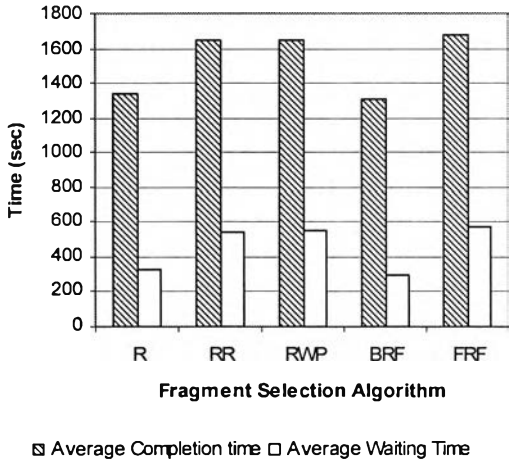
4.2 Performance for Fragments with Different Sizes and Same Numbers of Replicas

The experiments in this section are similar to those in Section 4.1, except that each fragment has the same number of replicas in these experiments. They are designed to study the performance of the proposed strategies with various replication degrees. For each experiment, every fragment has the same number of replicas, varying from 1 to 5. Each fragment is replicated to the five servers with 1 replica, 2 replicas, 3 replicas, 4 replicas and 5 replicas.

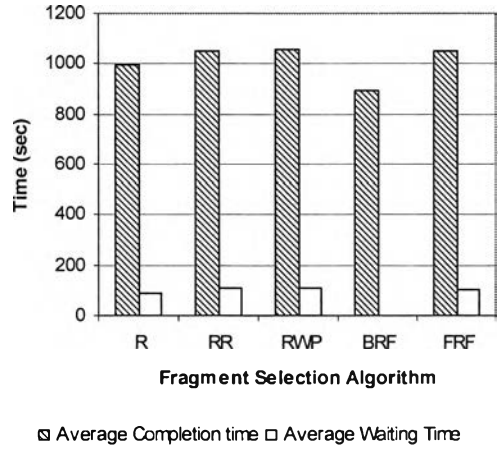
From the both grid environments, the experiment shows that the performances of all five algorithms are not significantly different if the number of replica is 1, 4 and 5. When the number of replica is 2, comparing to Random algorithm, Biggest-size-first algorithm yields marginally better average completion time approximately in the first environment and approximately 20% in the second environment. When the number of replica is 3, Biggest-size-first algorithm yields the best performance in term of average completion time which is approximately 10% better than Random algorithm in both grid environments as shown in Figure 18 - 19. Random, Round-robin and Fewest-replicas-first algorithms perform worst.



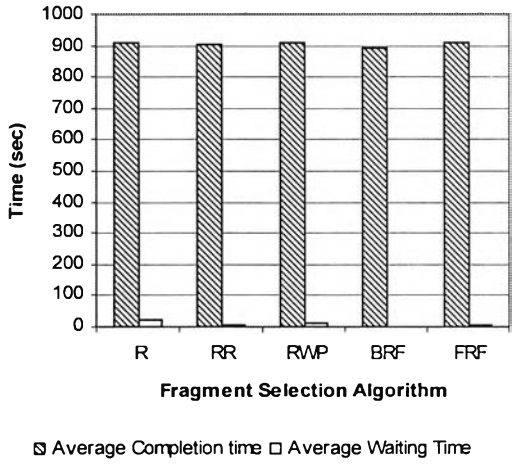
(a) 1 replica



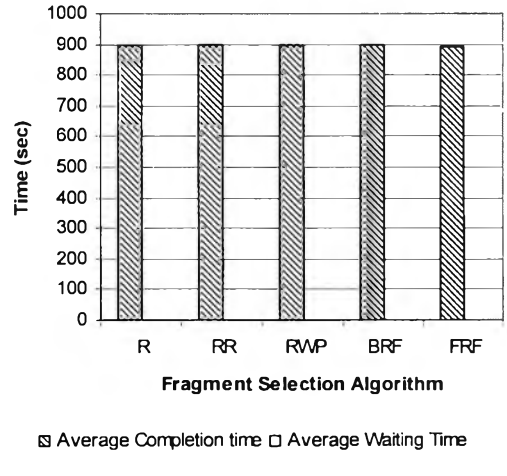
(b) 2 replicas



(c) 3 replicas



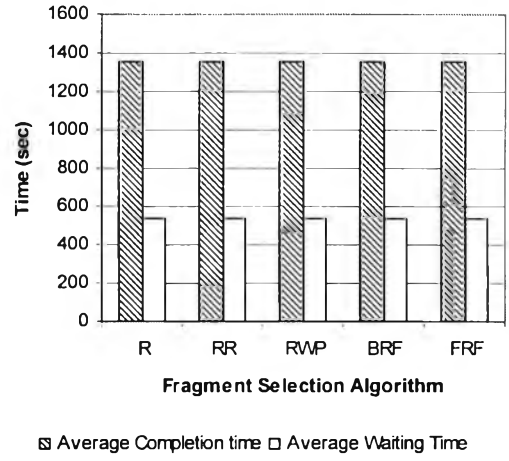
(d) 4 replicas



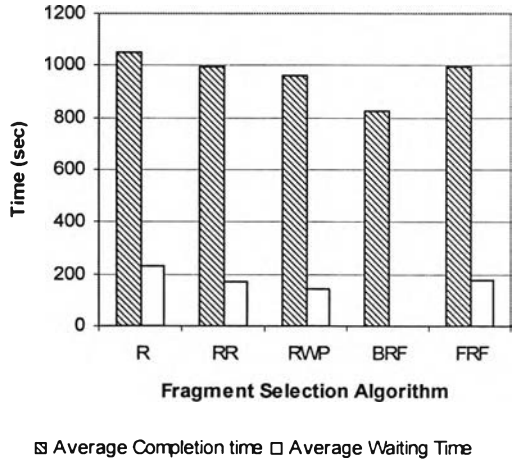
(e) 5 replicas

R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

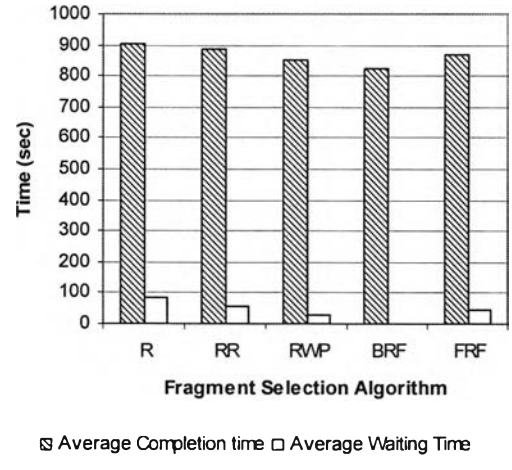
Figure 18: Average completion time in grid that has different bandwidths



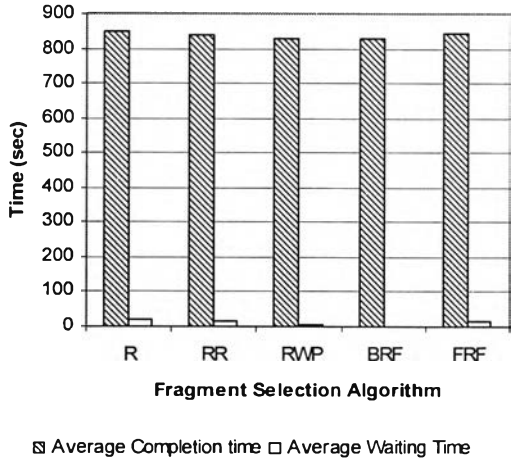
(a) 1 replica



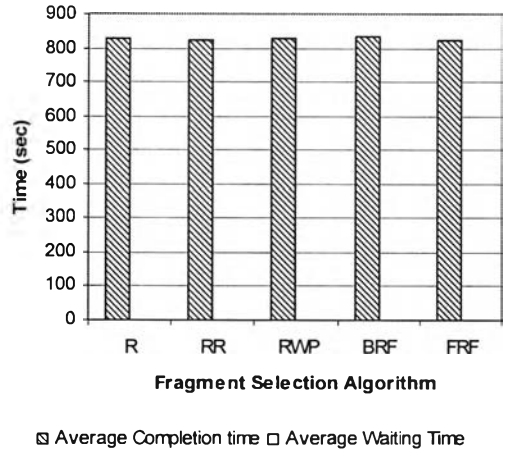
(b) 2 replicas



(c) 3 replicas



(d) 4 replicas



(e) 5 replicas

R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

Figure 19: Average completion time in grid that has same bandwidth

Fewest-replicas-first algorithm cannot be used in the situation when every fragment is equally replicated. Fewest-replicas-first algorithm selects fragments whose number of replica is fewest. In this situation, the number of replicas is the same, and the co-allocator selects the replica randomly.

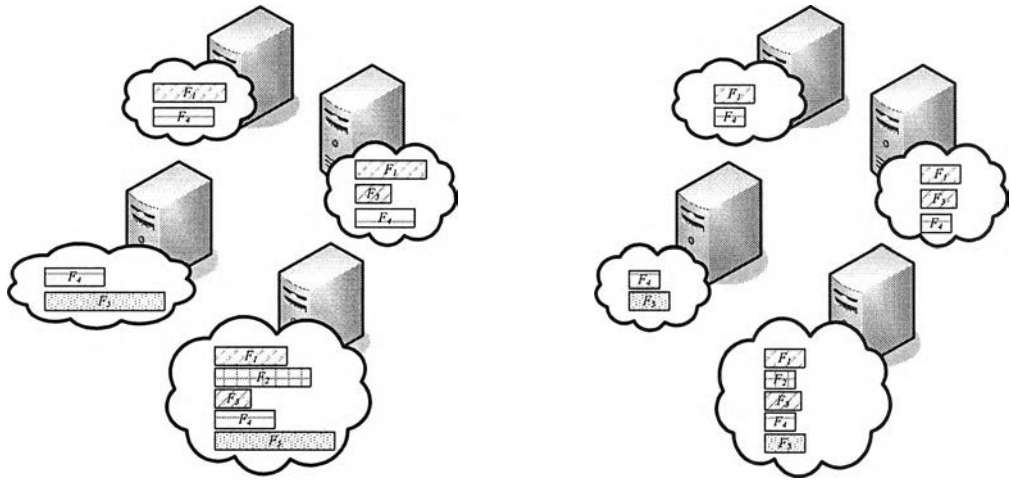
From this experiment, the average completion time is reduced when the number of replicas is increased in both grid environments. This means that grid has more replicas of fragment to transfer for each server. For example, when a few fragments are replicated on a server, the server is idle when these fragments are completely transferred to the client and other servers still have more fragments of data to be transferred. If the number of replicas is increased, the server can help the other transfer remaining fragments. So, the completion time is reduced.

When each server has one replica, the co-allocator has no choice to select the fragment, so the fragment located in the server is selected. This causes the same performance of all algorithms in both grid environments. When each server has five replicas, this can be considered that all servers have one complete data. The co-allocator can select any fragment at any server, so the average completion time of all algorithms is significantly different.

From this experiment, Biggest-remaining-first algorithm performs well for different fragment size. Biggest-remaining-first algorithm tries to enhance the collaboration of all servers at all time, so it sends a block from the biggest fragment first because it takes longer to complete the transmission. Another reason is that the algorithm tries to equalize the size of each fragment in all phrases. So, all fragments have approximately same size in all phrases. This means that all servers can help transfer fragments of data for all phrases.

For example, there are five fragments of data distributed on grid containing four servers as shown in Figure 5. At the beginning of the transmission, all four servers have fragments of data to be transferred to the client in parallel as shown in Figure 20 (a). If the fragments with biggest remaining fragment are sent first, the

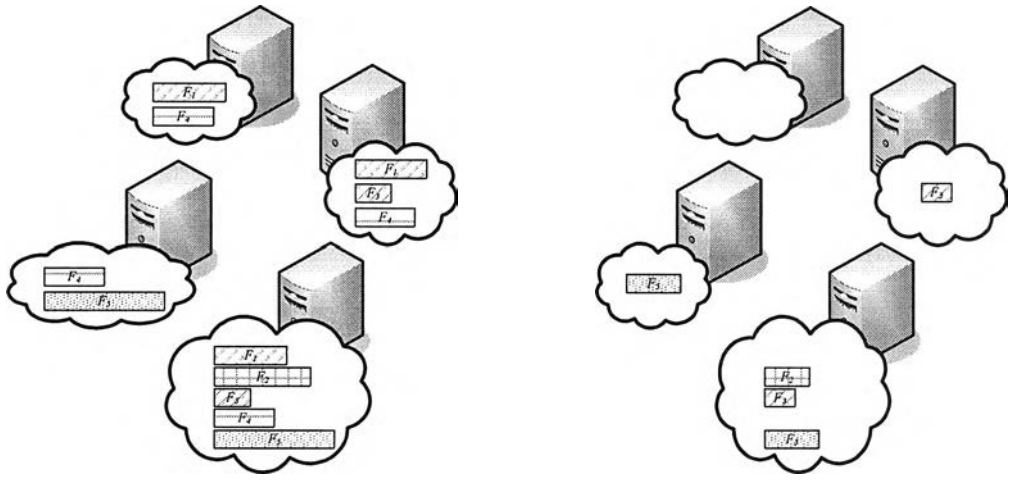
fragment size will be roughly equal in the later phrase. So, all four servers help transfer five fragments of data in parallel as shown in Figure 20 (b).



(a) The beginning of the transmission (b) The last phrase of the transmission

Figure 20: The beginning and last phrase of the transmission when using Biggest-remaining-first algorithm

On the other hand, if smallest remaining fragments are sent first, it takes short time to complete the transmission of these fragments. If a server has small fragments, the transmission of these fragments is finished first. Then the server is idle while the servers that have biggest unsent data left continue fragment transmission.



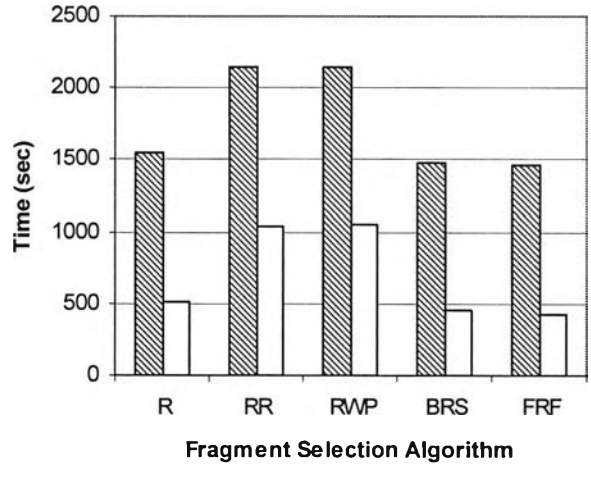
(a) The beginning of the transmission (b) The last phrase of the transmission

Figure 21: The beginning and last phrase of the transmission when not using Biggest-remaining-first algorithm

For example, shown in Figure 21 if fragments F_1 and F_4 are first completely transferred to the client, one server is idle. Only three servers left to transfer fragments F_2, F_3, and F_5. The waiting time of one server is high, and the average completion is also high.

4.3 Performance for Fragments with Same Sizes and Different Numbers of Replicas

The experiments in this section are similar to those in Section 4.1, except that each fragment has the same size in these experiments. They are designed to study the performance of the proposed strategies with various fragment sizes.

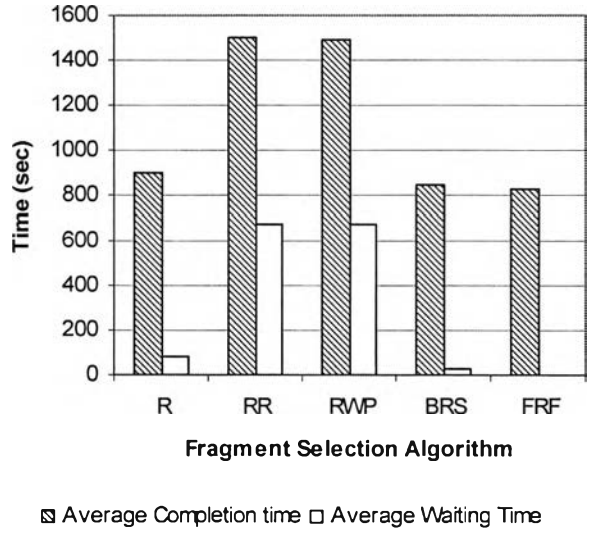


R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

Figure 22: Average completion in grid that has different bandwidths

From the experiment on both grid environments, Fewest-replicas-first algorithm yields the best average completion time as shown in Figure 22 - 22. Fewest-replicas-first algorithm performs better than Random algorithm approximately 5% in the first environment and 8% in the second environment. Comparing to Biggest-remaining-first algorithm, Fewest-replicas-first algorithm yields marginally better

average completion time in both grid environments. The Round robin algorithm and Random-with-weighted-probability algorithm perform worst.



R=Random, RR=Round-Robin, RWP=Random-with-Weighted-Probability
BRF=Biggest-Remaining-First, FRF=Fewest-Replicas-First

Figure 23: Average completion time in grid that has same bandwidth

The reason contributing to the performance of the experiment is the same as shown in Section 4.1 - 4.2.

4.4 The Effect of Variance of Transmission Rate on the Proposed Algorithms

The experiments in this section are similar to those in Section 4.1. They are designed to study the effect of the change of available bandwidth.

The average completion time for different percentage of coefficient of variation are shown in Figure 24. For different percentages of coefficient of variation, the change in average completion time is relatively small for all algorithms.

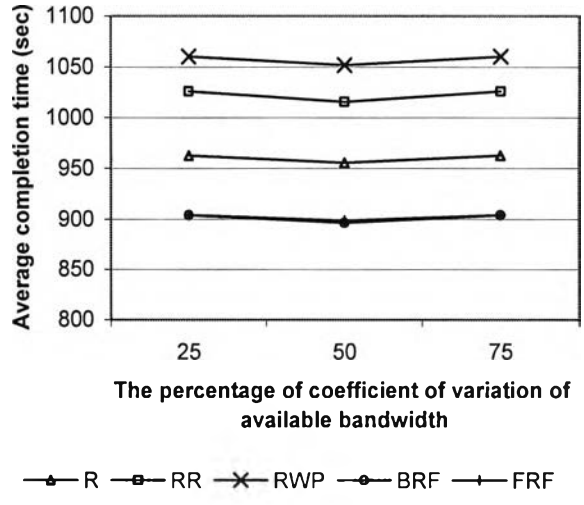


Figure 24: Average completion time in different percentages of coefficient of variation of transmission rate

The insensitivity of the variance of bandwidth results from the behavior of dynamic co-allocation. All algorithms divide data into many blocks. When a server completes a block transmission, it requests for the next block. Consequently, when the transmission is slow for a server, that slow server gets fewer blocks to transfer. On the other hand, a server can transfer data faster; the server gets more blocks to transfer. That is, the allocation adjusts automatically according to the effective transfer rate. As a result, the variation of transmission rate only has small impact on all algorithms.

Next chapter describes the conclusions of this thesis and future works.