CHAPTER III

Proposed Co-allocation Strategies

This chapter presents grid architecture for co-allocation assumed for the study in this thesis. It includes the responsibilities of client, server and co-allocator in the co-allocation scheme. Then, the next section presents algorithms for fragment selection which is an important part of the co-allocation for fragmented replicas. Five algorithms - Random, Round-robin, Random-with-weighted-probability, Biggest-remaining-first and Fewest-replicas-first algorithms - are studied

3.1 Grid Architecture for Co-allocation

The architecture for co-allocation in this study is adapted from the architecture of the dynamic co-allocation scheme as shown in Figure 1. A grid is composed of many servers connected via network. One server is chosen as a co-allocator which connects to a client. The client wants to download a *dataset* from servers. Other servers S_1, S_2, S_3, ..., S_n contain replicas of fragments of the dataset.

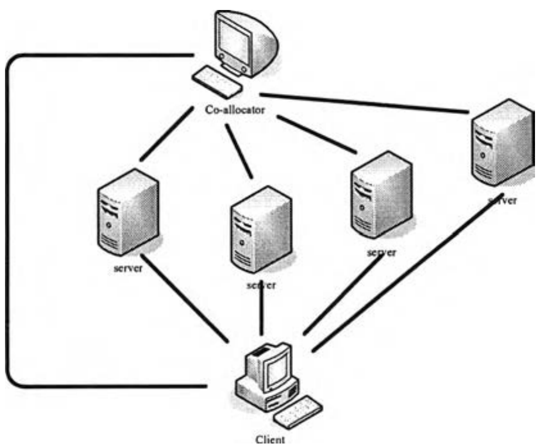

Figure 1: Grid architecture

The dataset is composed of m non-overlapping *fragments* F_1, F_2, F_3, ..., and F_m, whose sizes are denoted by $|F_1|$, $|F_2|$, $|F_3|$, ..., and $|F_m|$ respectively. All fragments are not necessarily of the same size. Each fragment is divided into blocks of k MB. Then, a fragment F_i is divided into $w = \lceil |F_i| / k \rceil$ blocks. The size of all but the last block of a fragment is k MB, and the size of the last block is $|F_i|$ mod k, where $i = 1, 2, 3, ..., m$.

Replicas of these fragments are stored in servers. Each server can replicate any number of fragments, and does not necessarily have replicas of every fragment.

Figure 2 shows a dataset consisting of five fragments, F_1, F_2, F_3, F_4 and F_5. The fragment F_1 is replicated to server S_1, S_2 and S_3, the fragment F_2 to only server S_3, the fragment F_3 to server S_2 and S_3, the fragment F_4 to all servers and the fragment F_5 to server S_3 and S_4.

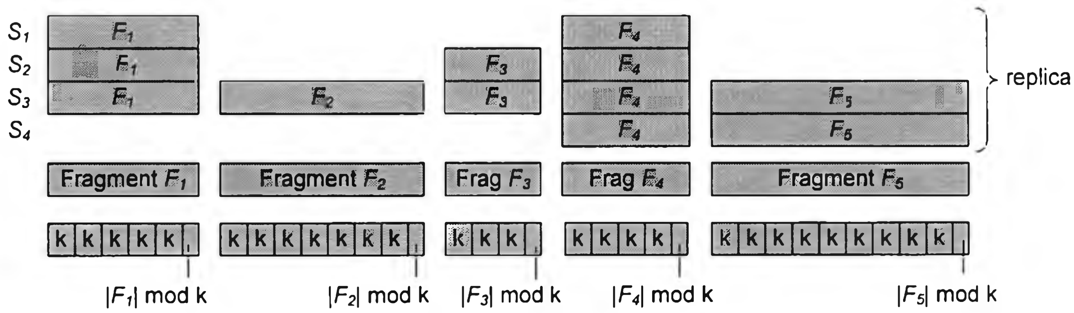

Figure 2: The replication of fragments among servers

3.1.1 Client

When the client wants to download a dataset, it sends a request to the co-allocator. The client then waits for blocks of data from servers, which transmit blocks according to the assignments from the co-allocator. Once the client gets a block, it sends a message to the co-allocator to report the completion of the block transmission, as shown in Figure 3.

The client:

Sent a request to download dataset

ON receive the block B from the server S_i

Send a message to the co-allocator to report the completion of transmission of block B from the server S_i

Figure 3: Function of the client

3.1.2 Servers

A server waits for the assignment from the co-allocator. After the assignment is received, the server transfers the assigned block to the client, as shown in Figure 4

Server:

ON receive the assignment to send a block *B* to the client

Transfer the block *B* to the client

Figure 4: Function of the server

3.1.3 Co-allocator

The co-allocator knows the location of all replicas. Once the co-allocator gets the request, it uses this information to assign servers to send blocks of the replicas.

The co-allocator decides, for each server, which fragments, among all fragments that are available in the server, should be assigned for transmission at that time. Once the co-allocator assigns the fragment to the server, the server chooses a block in the assigned fragment to be transferred to the client. From Figure 5, the fragments F_1 and F_4 are replicated in S_1. The co-allocator can assign a block from the replicas of either F_1 or F_4 to the server S_1 according to fragment selection algorithms described in Section 3.2. On server S_2, the fragments F_1, F_3 and F_4, are replicated. As a result, the co-allocator assigns a block from replicas of F_1, F_3 or F_4.

When a client wants to download fragments of data, it sends a request to the co-allocator. The co-allocator waits for a request sent by a client to start transmission. When the co-allocator gets the request, it divides all fragments into small blocks and set the state of blocks to *initial*. It chooses a fragment for each server and then chooses a block from the given fragment. The co-allocator assigns the chosen block to server S_i for transmission.

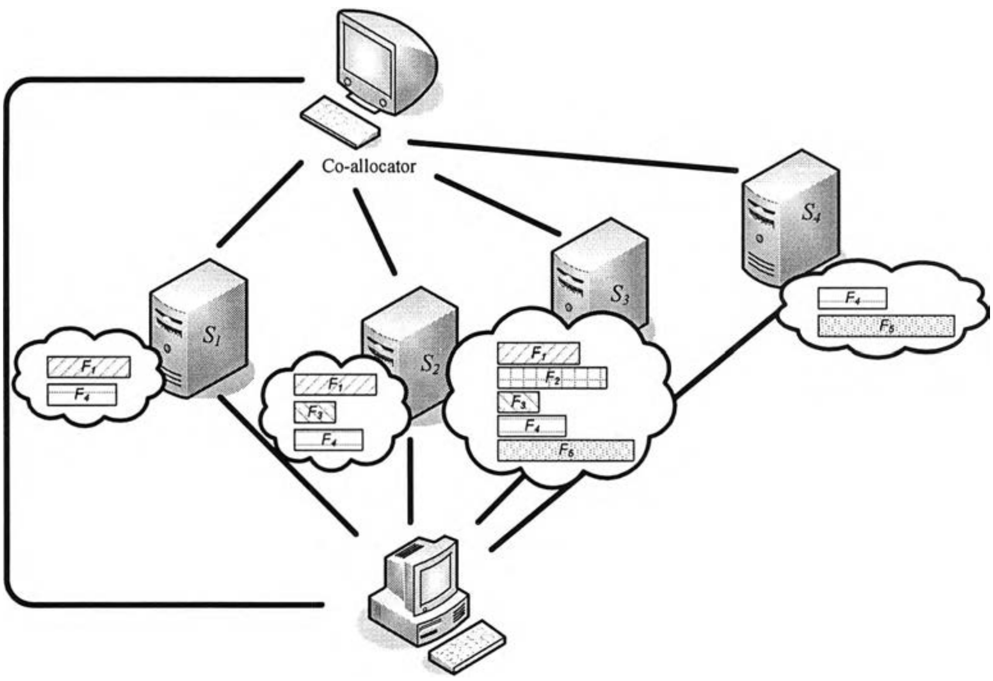

Figure 5: Example of fragment placement on servers

Once a server finishes transferring an assigned block, it waits for the next block assignment from the co-allocator. When the co-allocator receives the message from the client reporting the completion of a block transfer from the server S_i, the co-allocator chooses a replica among all replicas on the server, S_i, using function choose_replica, and assigns a block of the chosen replica to the server S_i, as shown in Figure 6.

The strategies for choosing a replica of a fragment, which are studied in this thesis, will be presented in Section 3.2.

```
Co-allocator:
ON receive a client's request
        Divide all fragments into blocks and set the state of all blocks to initial.
        FOR each server i among all n servers
                /*Choose an appropriate fragment for server i */
                F_a = choose_fragment(server i).
                /* Choose a block in the replica of F_a */
                B = choose_block(F_a).
                Assign the block B to server i.
        ENDFOR

ON receive completion report of block B by the server i, from the client
        Change the state of the block B to completed.
        IF there is an uncompleted block available on server i
                /*Choose an appropriate fragment for server i */
                F_a = choose_fragment(server i).
                /* Choose a block in fragment F_a */
                B = choose_block(F_a).
                Assign the block B to server i.
        ENDIF

FUNCTION choose_block(fragment F)
        IF there is a block B of the fragment F in the state initial
                Change the state of B to assigned.
                RETURN B
        ELSEIF there is a block B of the fragment F in the state assigned.
                RETURN B
        ENDIF
```

Figure 6: Function of the co-allocator

After the replica is chosen, the co-allocator chooses a block of the chosen replica, using the function choose_block shown in Figure 6, to be transmitted based on the state of the blocks. A block in each replica is in one of the three possible states – *initial*, *assigned*, and *completed*. In the beginning, each block is in the *initial* state. The state of the block is changed to *assigned* when the co-allocator assigns the block to a server, and changed to *completed* when the co-allocator receives a message from the client that the block is successfully downloaded a from the server, as shown in Figure 7.

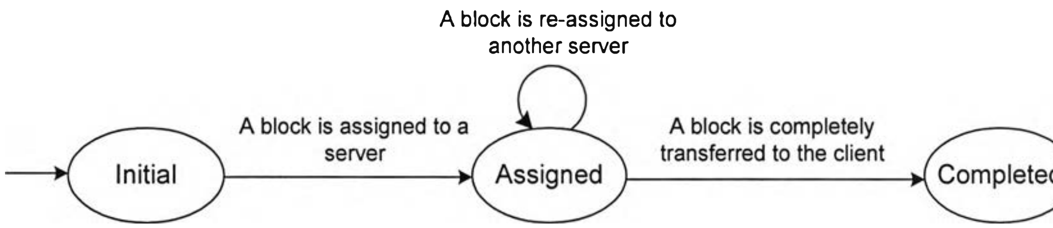

Figure 7: State transition of a block

Function choose_block(fragment F_a) selects a block of a given fragment. It checks all blocks of the given replica. If there is a block in the *initial* state, the co-allocator assigns a block in the *initial* state first. Otherwise, the co-allocator finds the block in the *assigned* state and re-assigns this block to a server. The reassignment is useful when a block gets delay by broken or congested links. However, if the client receives more than one copy of the same block, the co-allocator keeps the first block and discards the rest. If there is no block in the *assigned* or *initial* state, i.e., all blocks are in the *completed* state, the fragment will not be further considered for transmission. This process is repeated until all fragments are transmitted.

For the co-allocator to assign a block to a server, it must first choose an unfinished fragment for the transmission, shown as the function choose_fragment in the algorithm in Figure 6. The algorithms to choose fragment are described in the next section.

3.2 Co-allocation with Fragment Selections

This section describes fragment selections studied in this thesis. Random and Round-robin algorithms presented in Section 3.2.1 and 3.2.2 are used as baselines for the comparison to Random-with-weighted-probability, Biggest-remaining-first and Fewest-replicas-first algorithms presented in Section 3.2.3 - 3.2.5.

3.2.1 Random Algorithm

This algorithm is used as one of baseline algorithm in this study because, using no information about grid or replicas, the assignment of fragments to servers is chosen at random with equal probability among all fragments. For this algorithm, when the co-allocator assigns a fragment to server S_i, the co-allocator randomly chooses one fragment, among all fragments have not been completely transmitted, with uniform probability. If the server S_i has a replica of the chosen fragment, the co-allocator assigns the server S_i to transfer the chosen fragment, if not, the chosen fragment will be considered for the next assignment so that the fragment does not lose its chance. The co-allocator continues choosing a fragment until it gets one with a replica located in the server. The next time the co-allocator chooses a fragment, it picks a fragment that is previous chosen but cannot be assigned before choosing one at random.

To implement Random algorithm, a linked list is used to store the chosen fragments which cannot be assigned because its replica is not stored on the server. Once the server S_i requests for a fragment, the co-allocator assigns the first fragment in the linked list which is replicated on the server S_i and removes it from the linked list. However, if the co-allocator cannot find a fragment located in the server, it randomly chooses a fragment as described earlier. Every chosen fragment which cannot be assigned is kept in the linked list for next assignments. This algorithm is presented in Figure 8.

Consider an example of fragment placement on server in Figure 5. Figure 9 shows the linked list at different time. At t_0 the linked list is empty and the co-allocator finds a fragment from the server S_1. At time t_1, the co-allocator gets the message that server S_1 finishes transferring the assigned block. Suppose the co-allocator chooses the fragment F_2 but is not in the server S_1. The co-allocator keeps the fragment F_2 in the linked list, as shown in Figure 9, and randomly re-chooses another fragment. Suppose the fragment F_4 which is in the server S_1 is chosen. Then, the co-allocator assigns the fragment F_4 to the server S_1 to be transmitted.

```
FUNCTION choose_fragment(server i)
        IF      there is a fragment F_a in the linked list where F_a is replicated at server i
                Remove F_a from the linked list
                RETURN F_a
        ELSE
                Randomly choose a fragment F_a from all fragments which need to be
transferred with uniform probability
                WHILE F_a is not in server S_i
                        Append F_a to the linked list
                        Randomly choose a fragment F_a from all fragments which need to be
transferred with uniform probability
                ENDWHILE
                RETURN F_a
        ENDIF
```

Figure 8: Random algorithm

At time t_2, the co-allocator finds a fragment for the server S_2. It starts by searching for the entries in the linked list. It finds only the fragment F_2 which is not located in the server S_2. So, the co-allocator randomly picks a fragment, say F_5. But the fragment F_5 is not located in the server S_2, and the fragment F_5 is added to the linked list as shown in Figure 9. The co-allocator re-chooses a fragment, say F_4. The co-allocator assigns the fragment F_4 to the server S_2.

At time t_3, the co-allocator finds a fragment for the server S_1. It searches for the entries in the linked list, and there is no fragment in the linked list which is located in the server S_1. So, the co-allocator randomly chooses a fragment, say F_2. Since the fragment F_2 is not located in the server S_1, it is stored in the linked list as shown in Figure 9. The co-allocator randomly re-chooses a fragment, say F_1. Since the fragment F_1 is in the server S_1, the co-allocator assigns the fragment F_1 to the server S_1.

At time t_4, the co-allocator finds a fragment for the server S_3. It searches in the linked list and finds the fragment F_2 which is replicated to the server S_3. So, the co-allocator removes the fragment F_2 from the linked list, as shown in Figure 9, and assigns the fragment F_2 to the server S_3.

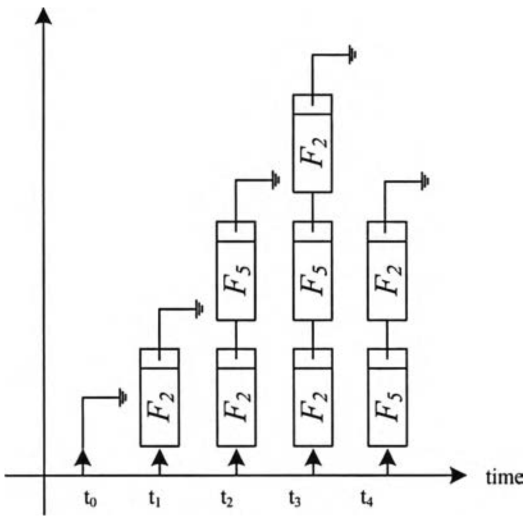

Figure 9: Linked list for Random algorithm

3.2.2 Round-robin Algorithm

Like Random algorithm, Round-robin algorithm is used as a baseline for comparing different strategies, but this algorithm focuses on fragments replicated locally. For each server, every replica located in the server has an equal chance to be selected.

When the co-allocator receives a message from the client that block transmission from a server is completed, the co-allocator selects a fragment from all fragments replicated on the server in turn. Unlike Random algorithm, Round-robin algorithm chooses only fragments stored locally. As a result, a fragment with more replicas has higher chance to be transmitted.

For the replication shown in Figure 10, the server S_1 has replicas of the fragments F_1 and F_4. When the co-allocator assigns fragments to the server S_1, the fragments F_1 and F_4 are assigned in turn. If the whole fragment is completely transferred to the client, it will no longer be considered for assignment. For example, if the fragment F_1 is completely downloaded to the client, the co-allocator always selects the fragment F_4 for the server S_1.

Similarly, in the server S_2, the fragments F_1, F_3 and F_4 are replicated in server S_2. The co-allocator selects the fragment F_1 first and then F_3 and F_4, i.e., the fragments F_1, F_3 and F_4 are assigned in turn.

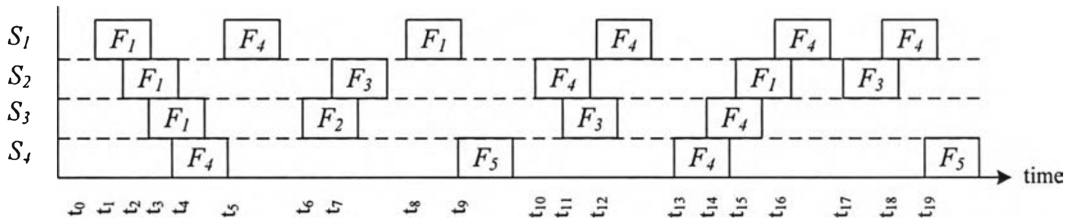

Figure 10: Fragments to be assigned at time t for Round-robin algorithm

Figure 10 shows the fragment assigned to each server at different point of time. Suppose the co-allocator finds a fragment to be assigned to the servers. At t_1, the co-allocator assigns the fragment F_1 for the server S_1. At t_2 to t_4, the co-allocator assigns the fragments F_1 to both the servers S_2 and S_3, and F_4 to the server S_4. At t_5, the co-allocator gets the message that the server S_1 finishes transferring the assigned block; it assigns the fragment F_4 because its last assignment is the fragment F_1. So, at t_8, when the server S_1 requests the next block, the co-allocator assigns the fragment F_1.

The server S_4 has two fragments, the fragments F_4 and F_5. When the server S_4 is assigned a block, it gets blocks from the fragments F_4 and F_5 in turn as shown in t_4, t_9, t_{13} and t_{19}.

3.2.3 Random-with-weighted-probability Algorithm

As described earlier, Random algorithm selects each fragment with uniform probability. Like Random algorithm, Random-with-weighted-probability algorithm selects fragment randomly, but the probability for choosing each fragment is weighted by its size.

For each server, the co-allocator chooses a fragment, among all replicas stored at the server, at random with the probability proportional to the size of fragment replicated on that server. That is, a bigger fragment is chosen more often than a smaller fragment. When a fragment replicated on the server is completely transferred, the probability is changed because a completed fragment is no longer considered.

For example, fragments F_1, F_2, F_3, F_4 and F_5 are distributed on a grid. The ratio of $|F_1|$: $|F_2|$: $|F_3|$: $|F_4|$: $|F_5|$ is 6: 8: 4: 5: 10. So, the co-allocator assigns blocks from the fragment F_5 more often than that from the fragment F_2 and blocks from the fragment F_2 more often than that from the fragment F_1 and the block from the fragment F_1 more often than that from the fragment F_4 and the block from the fragment F_4 more often than that from the fragment F_3. If a server has replicas of all these five fragments, the probabilities to choose the fragments F_1, F_2, F_3, F_4 and F_5 are $\frac{6}{33}$, $\frac{8}{33}$, $\frac{4}{33}$, $\frac{5}{33}$ and $\frac{10}{33}$, respectively.

If another server has replicas of the fragments F_1 and F_4, the probabilities of F_1 and F_4 to be chosen are $\frac{6}{11}$ and $\frac{5}{11}$, respectively.

Random-with-weighted-probability algorithm uses the original fragment size to determine the fragment selection. This information is static and can be defined before data transmission. To select the fragment based on the original fragment size might not reflect the real-time situation because the fragment size has been changed all the time during the transmission.

3.2.4 Biggest-remaining-first Algorithm

As described in the previous section, the probability of choosing each fragment is static and does not reflect the actual workload. On the other hand, Biggest-remaining-first algorithm uses the size of remaining replica as the factor to choose a fragment. This helps the system adjust to select fragment. The bigger fragment takes longer to transfer comparing to a smaller one. Later, this might cause that the server transfers only the bigger fragment while the small one is completed and other servers must wait for this server. To balance the bigger and smaller fragment transmission, the co-allocator should select the fragment that has the bigger size to be sent first.

For this strategy, the co-allocator chooses a fragment, among all replicas stored at the server, by considering the size of the remaining fragment. A fragment which has the biggest amount of unsent data is sent first. Thus, the size of unsent data needs to be updated when a block of the fragment is completely transferred to the client. When the co-allocator finds a fragment to assign to a server, the co-allocator compares the remaining size of each fragment and chooses the fragment that has the biggest amount of remaining data. If there are more than one replica has the same amount of remaining data which is the biggest, the co-allocator randomly selects from one of those fragments.

Figure 11 shows the fragment selection using Biggest-remaining-first algorithm. When fragments are shown in dash line, it means that the fragments are not replicated on the server. At time t_0, the co-allocator gets the message that the server S_1 finishes transferring the assigned block and finds a fragment to assign to the server S_1. The co-allocator checks the remaining fragment size of the fragments F_1 and F_4 which are replicated on the server S_1 as shown in Figure 11. Because the fragment F_1 has bigger remaining amount of data, the co-allocator assigns the fragment F_1 to the server S_1. Once the client gets the block of the fragment F_1, the remaining size of the fragment F_1 is updated.

At time t_1, the co-allocator finds a fragment for the server S_2. It checks the remaining amount of unsent data of the fragments F_1, F_3 and F_4 which are replicated on the server S_2. The fragments F_1 and F_3 have the same biggest remaining amount of unsent data among three fragments, so the co-allocator selects randomly from these fragments and the fragment F_1 is selected. Suppose the block transmission of the fragment F_1 is done between time t_1 and t_2, so the size of remaining data of the fragment F_1 is updated.

At time t_2, the co-allocator finds a fragment for the server S_3. It selects the fragment F_5 because it has the biggest amount of remaining data. The block transmission of the fragment F_5 is completed before time t_3, so the size of remaining data of the fragment F_5 is updated.

Similarly, at time t_3, the co-allocator selects the fragment F_5, and the fragment size of remaining data of the fragment F_5 is updated before time t_4.

At time t_5, the size of remaining data of the fragment F_5 still remains because it is not yet completed transmission. However, this is not considered because the fragment F_5 is not replicated to the server S_1. The remaining fragment size of the fragment F_5 is updated before time t_6 when the block is completed transmission.

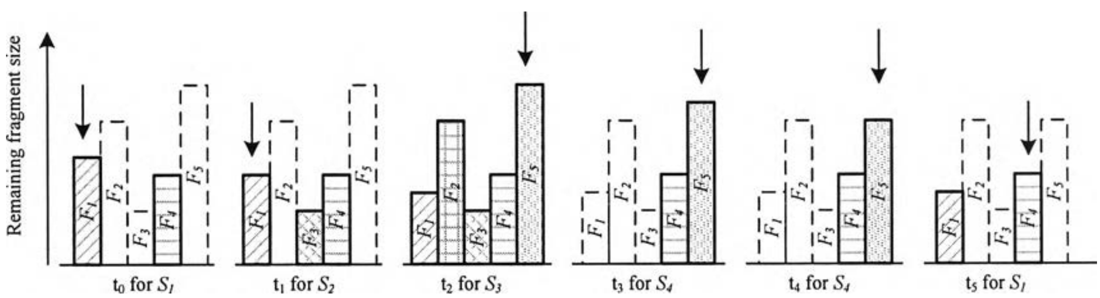

Figure 11: Remaining size of fragments for Biggest-remaining-first algorithm

3.2.5 Fewest-replicas-first Algorithm

A fragment that has been replicated to fewer servers has less chance to be selected and a fragment that has more replicas can be transmitted to the client in the short time. If we use Random algorithm or Round-robin algorithm, each fragment has equal chance to be selected; the fragment that has fewer replicas might be transmitted after other fragments are completely transferred to the client. This causes other servers wait for this server. To solve this problem, this algorithm considers a number of replicas as a major factor to select the fragment for servers.

For this algorithm, the co-allocator chooses a fragment, among all replicas stored at the server, that has fewest replicas first. If one fragment is replicated to fewer servers than other fragments, the fragment should be transferred first. Once a fragment is completely transferred, the co-allocator no longer considers that fragment and chooses another fragment in the server that has next fewest replicas. If there is more than one fragment that has the same number of fewest replicas, the co-allocator will randomly pick from those.

Figure 12 shows the number of replica of each fragment. At the time t_0, the server S_1 has replicas of the fragments F_1 and F_4. The fragment F_1 has three replicas, and the fragment F_4 has four replicas. So, the co-allocator selects the fragment F_1 to be assigned to the server S_1.

At the time t_1, the co-allocator finds a fragment to be assigned to the server S_2. It checks for number of replica of the fragments F_1, F_3 and F_4 as shown in Figure 12. So, the co-allocator selects the fragment F_3 for the server S_2.

At the time t_2, the co-allocator finds a fragment to be assigned to the server S_3. There are five fragments replicated to the server S_3, but the fragment F_3 is completely transferred to the client. So, the co-allocator considers only other four fragments and it selects fragment F_2 because it has fewest replicas.

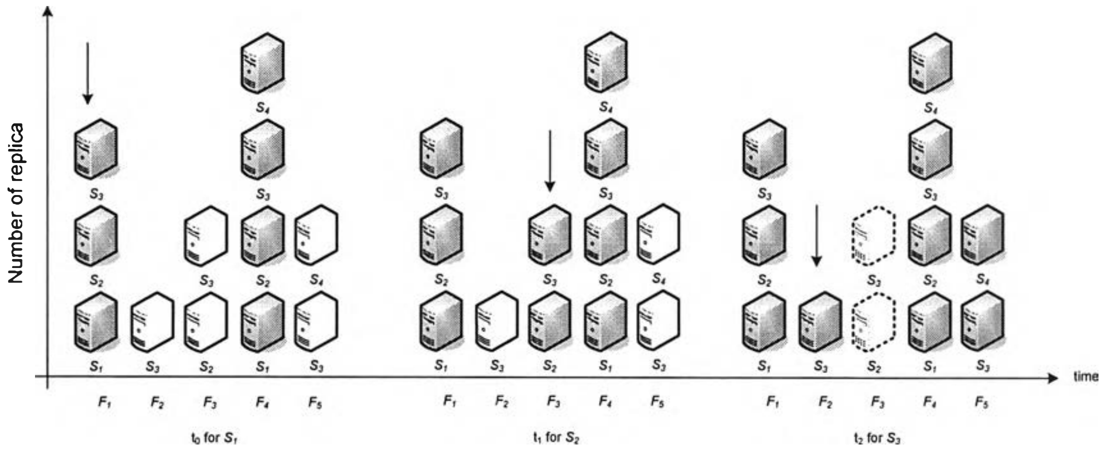

Figure 12: Number of replica at time t for Fewest-replicas-first algorithm

In the next chapter, a group of experiments is described to evaluate the performance of our proposed algorithms shown above.