# CHAPTER II

# Related Works

In data grid environments, there are many servers which are different in network connectivity, storage system architectures, and system load. When a client wants to use data which is not located in the local server, the system locates the requested data on other servers and replicates the data to the client. However, the data used in data grids is massive. To transfer or replicate the data from one server to another can take a long time. Moreover, some applications have time constraint to finish. Improvement of download speed can make applications processed sooner. Data replication, replica selection and co-allocation strategies are used to improve download speed.

This chapter describes strategies for data replication, replica selection and co-allocation in Section 2.1, 2.2, and 2.3 respectively. Then, the concept of fragmented replica, which is used to reduce the storage requirement, is described in Section 2.4.

## 2.1 Data replication strategies

A data replication strategy, or replica management [11], is a process to replicate an original dataset from one server to other servers or to delete a replica from a server [7]. To create a replica, a grid has to select an appropriate location to store the replica. Sometimes a dataset is required for a server, but there is insufficient storage for the dataset in the server. The replica manager needs to delete some existing replicas to make space for the new dataset. The replica manager must also maintain a replica catalog containing the information required for mapping a logical collection to a particular physical instance of that collection [12].

Data replication can increase the system reliability and can reduce data access time. When a client wants to access a dataset, it can choose from any one of the replicas of the dataset. Sometimes the server, which stores the original copy of the dataset, is not available because of broken connection. The system can use any replicas of the dataset, and thus increases system reliability. Furthermore, a client can choose to access a replica of a dataset in order to reduce data access time [11].

There is a research proposing a data replication strategy which combines data replication algorithm with job scheduling policy [13]. When a client requests for remote data, the replica manager fragments the data into small pieces. Then, it selects the best replica of each piece, based on some criteria, e.g., neighborhood, network bandwidth, and availability. Pieces of data are replicated from those sources and are assembled together into the original data. This strategy allows the system to fully utilize network bandwidth due to multiple-source replication. In addition, this strategy improves system efficiency and reliability.

## 2.2 Replica selection strategies

When a client requests a dataset that is not located in the local server, a grid system can transfer the dataset from any server storing the replica of the dataset. Normally, a grid system selects a server located near the client to reduce data transfer time [14]. However, it can take a long time to transfer data if the server is slow, or has slow connection. Replica selection is proposed to choose an appropriate replica for each situation. A replica selection strategy chooses a replica, among all available replicas that will provide an application with data access characteristics that optimize a desired performance criterion such as speed, cost, or security [15]. One of common criteria for replica selection is replica access time, which is contributed from many factors, e.g., disk access time, network traffic and the workload of the replica host [8]. The replica manager must maintain this information for replica selection.

A high-level replica selection service which considers the replica location and user preferences to select replica is proposed in [7]. The replica selection uses decentralized storage broker in each client, instead of a centralized manager and uses the statistical information from an information service such as the Network Weather Service [16] to perform predictive analysis of the behavior and choose the appropriate replica.

Another replica selection using probabilistic technique is proposed in [8]. The ant algorithm [17] is a technique for solving combinatorial optimization problems. This algorithm imitates the behavior of ants in finding paths from their colony to food. This replica selection algorithm uses ant algorithm with the access time information, such as disk I/O transfer, network status, and replica host load, to select the replica. From the experiment, it shows that, compared to SimpleOptimiser algorithm [18], replica selection using ant algorithm performs better, in term of average access time, when there are many replicas in the grid. Moreover, this strategy consumes less network bandwidth and creates less storage site load.

## 2.3 Co-allocation strategies

Replica selection strategy selects one site with the best expected performance to transfer the data to the client based on the selection strategy. Only one selected site is responsible for the whole dataset transmission. The problem raised from this strategy is transfer reliability. If the selected site is down or the performance is dropped during transmission, the expected time to complete may change. Then the overall completion time might be high. Moreover, when several available replicas have almost the same network and disk throughput, choosing a slightly better replica while discarding all others is unreasonable [19].

To improve transfer reliability and transfer completion time, co-allocation concept is proposed to utilize the bandwidth from all servers to transfer replicas from different servers in parallel. Many workload allocation algorithms used in co-

allocation strategies try to equalize data transfer time among servers by assigning larger portions of data to faster servers and smaller portions to slower servers [10]. Others aim to use bandwidth of all servers to transfer data concurrently.

Workload allocation can be classified based on two criteria – adaptability and the use of history. Based on adaptability, workload allocation can be either static – the workload is allocated among servers only once before data transmission – or dynamic – workload assignment is dynamically adjusted during the transmission. Based on the use of history, some strategies use past performance to determine the workload allocation while others disregard the performance history. From the criteria stated, workload allocation can be categorized as follows.

### 2.3.1 Static allocation

For static allocation, workload is allocated only once before the data transmission begins, and then each server is assigned a portion of data to be transferred. Then, all servers transfer the assigned block simultaneously. Thus, the overhead in the transmission of each block can be minimized. The performance of this type of strategies depends on how balanced the workload is distributed. There are two variations of static allocation.

#### 2.3.1.1 Historyless static co-allocation

For historyless static co-allocation strategy, each available server is assigned one block of the same size. This strategy is called brute-force co-allocation. The study in [19, 20, 21] uses uniform co-allocation as a baseline for their study. The algorithm divides the file size equally among the available servers. For example, if a file size is $S$, and there are $n$ available servers. A data block of size $\frac{S}{n}$ is assigned to each server. So, this strategy utilizes the bandwidth from all servers. However, it ignores the difference of bandwidth among all available connections.

#### 2.3.1.2 History-based static co-allocation

Normally, history-based static co-allocation strategy aims to distribute the workload so that all servers take the same period of time for data transmission. As a result, servers with better transmission time are assigned more data while those with worse transmission time are assigned less data. For history-based static co-allocation strategies in [19, 20], the history of transfer rate is used to predict the present transfer rate, and data is assigned to each server proportional to the predicted transfer rate. The transfer rate prediction is done based on previous history of data transfers. The history-based static co-allocation strategy in [19] uses Network Weather Service [16] which is a distributed system to forecast short-term performance deliverable at the application level based on historical performance to predict the network throughput for each server. The strategy in [20] predicts transfer rate based on a previous history of GridFTP transfer [22]. GridFTP, part of the Globus Toolkit™, is widely used as a secure, high-performance data transfer protocol in Grids. For example, a file is divided into disjoint blocks, corresponding to $n$ servers. Each server, $S_i$, $1 \le i \le n$, has a predicted transfer rate of $B_i$ to client. So, total available transfer rate is $\sum_{i=1}^{n} B_i$. The block size for each server $S_i$ is $\frac{B_i}{\sum_{i=1}^{n} B_i} \times |F|$, where $|F|$ is the size of file. The performance of this strategy depends on how good the prediction is. Since the variance of transfer rate also indicates the reliability of the resources, *tuned conservative scheduling technique* [23] uses both predicted means and variances of network performance to determine the volume of data assigned to each server. In *one by one co-allocation* [24], data is divided into blocks of equal size, and each block is assigned to a server as a whole. In other words, the atomic unit of data is a block, not a byte. *One by one co-allocation* assigns a block of data to a server one by one so that the predicted completion time is minimal. An advantage of this approach is that faster servers need not wait for slower servers to finish their assigned transmission because the slower ones may get no assignment. The workload allocation is also based on the predicted transfer rate.

Since the network behavior cannot be precisely predicted, static workload allocation may not yield optimized performance [19]. If the transfer rate from a server is actually better than the predicted rate, this server is *underutilized*, i.e., it finishes the transmission before other servers. On the other hand, if the transfer rate from a server is worse than the predicted rate, this server is *stuck*, i.e., it still has data to be transmitted while some other servers are done.

### 2.3.2 Dynamic allocation

In dynamic allocation, data is divided into blocks and incrementally assigned to each server, and the workload allocation can be adjusted during the data transmission. This adjustment is usually based on the data transfer speed. However, some dynamic allocation strategies are historyless, and some are history-based.

#### 2.3.2.1 Historyless dynamic co-allocation

Similar historyless dynamic co-allocation is presented in [19, 20, 21]. This co-allocation strategy divides a dataset into disjoint and equal-sized blocks. Each server is assigned to transfer data one block at a time. When it finishes the transmission of a block, it requests another assignment. This is repeated until the whole dataset is transferred. Consequently, the faster servers deliver more blocks than the slower ones. The advantage of this approach is that the finish time among all servers is roughly equalized because the loading for each server is adjusted automatically without the explicit knowledge of the transmission rate. However, in contrast to the algorithm in [24], faster servers may have to wait for slower servers to transfer the final block.

To reduce this waiting period, the algorithm in [10] improves the co-allocation scheme by assigning the same block to other idle faster servers. Therefore, there may be duplication of blocks being transferred among servers. This can reduce the completion time of data transmission. In addition, this duplication transmission technique can protect the disruption of data transmission process when some network links are broken. Another approach using variable block size to reduce the waiting

period is to start by sending larger blocks and then gradually decrease the block size [25]. This reduces the total number of blocks to be transferred, and thus reduces the waiting time before transmission of the next block begin. In addition, the waiting time to wait for last block transmission is also reduced.

#### 2.3.2.2 History-based dynamic co-allocation

This approach uses the transfer rate to determine the amount of data allocated for each server so that all servers finish the data transfer at approximately the same time. As a result, the faster the server is, the more data it gets. The transfer completion time is improved by reducing the idle time faster servers spent waiting for the slowest server to finish transferring the last block.

The static allocation in [19] is modified to be more adaptive by dividing the dataset into fixed segments. Initially, every replica gets a portion of the first segment to start. When the last portion is already assigned and replicas are ready to start the next segment. The algorithm decides the portion of segment to be assigned to replicas. The proportion of workload assignment is adjusted according to the predicted transfer rate at each point of time. Two predictions of the transfer rate – NWS and last achieved transfer rate – are studied, and the results of this two predictions are similar.

Similar to the study in [19], recursive adjustment co-allocation [26] divides the dataset into segments, and allocates a segment among servers in each round. However, these segments are of different size, and a bigger segment is allocated in an earlier round. A segment of data is defined from the percentage of the remaining data. From the experimental result in [21], it shows that the percentage should be neither too large nor too small. So, the experiment in [26] uses 50% of the remaining data and sets the threshold segment size in order to avoid non-stop continue adjustment. A segment is then divided among all servers according to the predicted transmission rate and the size of the unfinished assignment of each server. The experimental result shows that this approach reduces the idle time spent waiting for the slowest server and decrease data transfer completion time.

An improvement of its historyless dynamic co-allocation in [20] is also presented. This strategy increases the number of data blocks assigned to a server if its transfer rate is high, and decreases the number of data blocks if the server's transfer rate is low. Among all servers, the most recent highest transfer rate is maintained. If the performance of a server exceeds the recent highest transfer rate, the server is assigned more blocks. If the performance is much lower than the recent highest transfer rate, the server is assigned fewer blocks. The experimental result shows that this technique performs better than static ones especially for the larger file sizes. Another improvement of the historyless dynamic co-allocation in [20] is abort and retransfer presented in [24]. This strategy aims to reduce the time faster servers spent waiting for the slowest server to finish data transmission. When the fastest server gets no more work, it allows a client to abort the slower server delivery and retransfer it from faster one. The client uses predicted transmission rate to estimate the time required for the fastest server to transfer a data block and the remaining time for the slowest server to complete the transmission. The assignment is reallocated if it is estimated that the faster server can finish the job earlier.

## 2.4 Fragmented data

In all works described earlier, a dataset is completely replicated, i.e., the whole dataset is copied for other servers. The concept of fragmented replica is proposed in [9]. A dataset is divided into fragments, and it is allowed to replicate some chosen fragments to each server, depending on storage availability and data authorization.

An important factor to be considered for data replication is the disk space. Data access can be vastly improved if all dataset are replicated on every server. However, it is not practical because of limited storage. It can be more effective to replicate some dataset as described in data replication strategies. On the other hand, it is possible to choose to replicate parts of a dataset on different servers.

Moreover, some applications have limited authorization for the client. For example, a publishing company stores its publications on its own servers. It may

allow some out-dated journals to be replicated on some library servers, but not allow current journals to be replicated to restrict access. Another reason the client cannot replicate some dataset is sensitivity. Some applications are not allowed to replicate the whole data; for example, biomedical applications deal with biological objects, medical image and medical data of patients in hospitals and medical institutions. The data in the application is highly sensitive. So, some data does not allow replicating due to sensitivity limitation [6].

Dynamic co-allocation algorithm proposed in [10] does not directly depend on the network performance prediction. For this strategy, a dataset is divided into blocks of equal size, and each server is assigned to transmit a block when it finishes the previous transmission. Thus, this algorithm dynamically adapts the workload assignment so that faster servers get more assignment. Recursive co-allocation is proposed to improve data transfer performance by modifying dynamic co-allocation [26]. This approach reduces data transfer completion time. Like dynamic co-allocation algorithm, one by one co-allocation [24] divides data into blocks of equal size. One by one co-allocation algorithm then assigns each block to the server that has minimum estimated completion time based on the network prediction. In contrast to dynamic co-allocation algorithm, one by one co-allocation relies on the network performance prediction.

This thesis proposed fragment selection algorithms by using dynamic co-allocation concept whose performance does not depend on the network performance prediction. The detail and example of proposed algorithms is presented in the next chapter.