Coordinated Co-allocator Model for Data Grid in Multi-sender Environment

R.S. Bhuvaneswaran, Yoshiaki Katayama, and Naohisa Takahashi

Department of Computer Science and Engineering,
Graduate School of Engineering, Nagoya Institute of Technology, Japan
{bhuvan, katayama, naohisa}@moss.elcom.nitech.ac.jp

Abstract. We propose a model, which simultaneously allocates a data block request to the multiple sites, termed as co-allocation, to enable parallel data transfer in a grid environment. The model comprises of co-allocator, monitor and control mechanisms. The co-allocation scheme adapts well to the highly inconsistent network performances of the sites concerned. The scheme initially obtains the bandwidth parameter from the monitor module to fix the partition size and the data transfer tasks are allocated onto the servers in duplication. The scheme is found to be tolerant despite the situation that the link to servers under consideration is broken or become idle. We used Globus toolkit for our framework and utilized the partial copy feature of GridFTP. We compared our schemes with the existing schemes and the results show notable improvement in overall completion time of data transfer.

Keywords: Data grid, co-allocation, parallel data transfer, GridFTP.

1 Introduction

Applications designed to execute on grids frequently require the simultaneous co-allocation of multiple resources in order to meet performance requirements [7][1][5]. For instance, several computers and network elements may be required in order to achieve real-time re-construction of experimental data, while a large numerical simulation may require simultaneous access to multiple supercomputers. Motivated by these concerns, several researchers [1][5][3][8] developed general resource management architecture for Grid environments, in which resource co-allocation is an integral component. Data store is one of the important resources and this paper deals about it. Several applications considered distributed data stores as resources [8][9]. Most of these data grid applications are executed simultaneously and access a large number of data files in a grid, termed as data grid. The data grid infrastructure is to integrate the data storage devices and data management service into the grid environment. Data grid consists of scattered computing and storage resources located dispersedly in the global network accessible to the users. These large sized data sets are replicated in more than one site for the better availability to the other nodes in

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 66 – 77, 2006.

a grid. For instance, in the multi tiered data grid architecture high energy physics experiments[9], replicated data stores are considered. Hence, any popular dataset is likely to have replicas located in multiple sites.

Instead of downloading the entire high volume dataset from a single server, the technique of downloading the data set parts from multiple servers in parallel that are consolidated at the client end, is of more theoretical and practical interest. This co-allocation of data transfer has alleviated most of the bottlenecks in downloading and improves the performance compared to the single server selection one. Many researchers realized this factor, discussed its advantages and proposed variety of techniques in different contexts [14][3][15]. We propose a model, which simultaneously allocates a data block request to the multiple sites, termed as co-allocation, to enable parallel data transfer in a grid environment. The model comprises of co-allocator, monitor and control mechanisms, naturally blended with feedback loop. The co-allocator scheme adapts well to the highly inconsistent network performances of the sites concerned.

We have experimented our schemes with Globus toolkit as the middleware and GridFTP. The results are compared with the existing approaches and the initial results outlast the existing ones performance. The rest of the paper is organized as follows: the problem is defined with the co-allocation model in section 2 and the related works in the same area are discussed in Section 3. We presented our proposed algorithm in Section 4 with assumptions and the experiments. Section 5 describes the analyses and Section 6 concludes the paper.

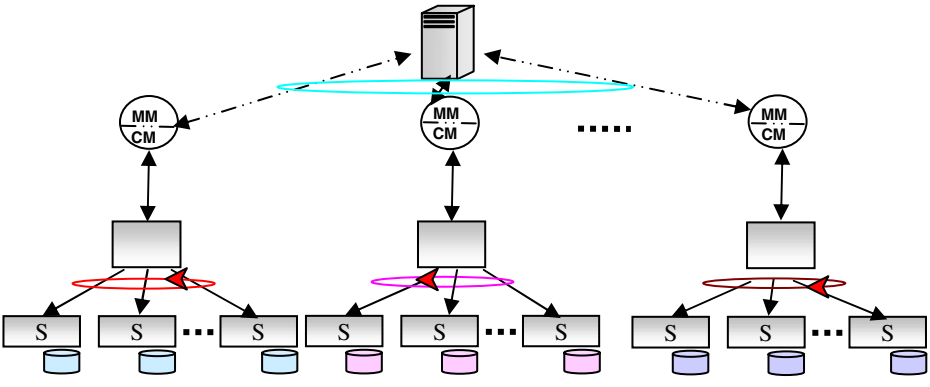


Fig. 1. Integrated Co-allocation Model of Data Grid with replicated multiservers

2 Co-allocation Model with Globus

The integrated coallocation model comprises of coallocation, monitoring and control components, discussed in detail in the subsequent sections.

2.1 Coallocation Mechanism and the Overall Architecture

The dynamic collacotor module integrated with monitor is shown in Fig.1. The model is presented from the client perspective. Each client has a coallocator agent (CM) and the periodical execution of monitor agent (MM).We used Globus Toolkit [11] which is an open source software toolkit used for building grid. The major component of a globus tool kit is GIS (Globus Information Service) which provide necessary information about the data stores in a grid. With help of GIS, the co-allocator adopted dynamic strategy to transfer data from multiple servers to the intended client. Fig. 2 depicts a single client point of view. The agent accepts the request from an application about the data solicited and its description is passed on to the co-allocator.

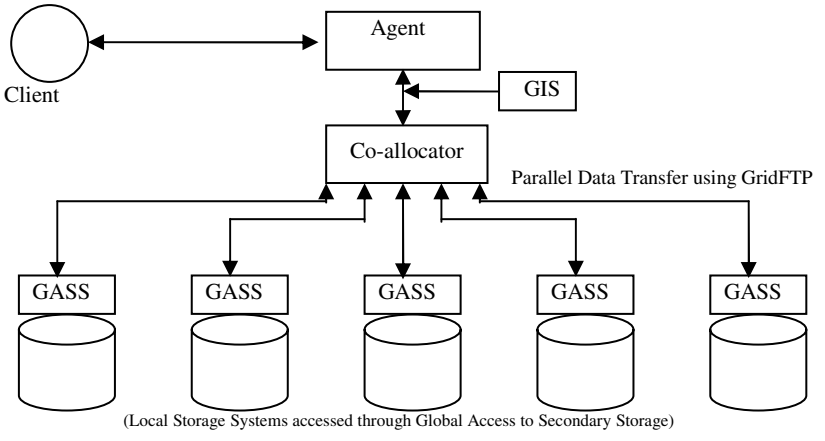


Fig. 2. The Co-allocation Model of Data Grid with replicated multiservers

The co-allocator, identifies multiple servers with the help of GIS, initiated the data transfer in parallel in parts. After all the parts of a data set are received, they will be assembled and give it back to the application through the agent. The file transfer is handled by GridFTP[11] service of a globus tool kit. We exploit the partial file transfer feature of GridFTP in our work and presenting a dynamic co-allocation strategy to enable the transfer of replicated data from multiple data servers, to a single client.

The outline of our strategy is spelled out as follows: The application (of client) requests the data with its description to co-allocator agent. Based on the information provided by GIS, the dataset is replicated and available in the servers scattered in the network. The dataset to be downloaded is divided into blocks. The co-allocator sends multiple requests to the server and the download process taken place in parallel. At the beginning, every server is assigned to transfer exactly one block respectively. When the requested block is received from a server, then, one of the yet unassigned blocks is assigned to the server. Co-allocator repeats this process until all the blocks of the dataset are assigned. Obviously, good performance servers transfer more

blocks than the slow servers. There may be duplication of blocks among the servers thereby reducing the waiting time from the slower servers and thus the fault tolerant factor can also be achieved.

2.2 Monitoring and Control Mechanisms

The allocation and configuration phases of the coallocation process result in the data transfer tasks on set of servers. During the data transfer, it is desirable to monitor and control the ensemble as a collective unit. The monitoring and control operations that we defined have this property. Monitoring operations allow a client program to receive notification when the resource set changes state. In addition to the obvious global state transitions of failure and termination, the complex failure modes encountered in Grid applications lead to a need to support and respond to individual process state transitions as well. Hence, the interface should allow for signaling operation (algorithm in Sec. 4) to the monitoring program, which can then act upon this transition in a manner that is appropriate for the coallocation. Similarly, control operations allow for the manipulation of the resource set as a whole. One required control operation is to check whether the received data is corrupted or not and the other one is killing the duplicate processes, which will be discussed in Sec. 4. Before we discuss about our strategy in detail, let us present the related works in this aspect.

3 Related Works

Few research works have been reported in the literature about parallel data transfer for the grid environment. They can be categorized into static and dynamic based on the allocation strategy. Once the allocation is made, it can never be changed during execution is termed as static, whereas, in dynamic allocations the allocations may be altered based on bandwidth or other performance criteria. t al [15] used past history of data to forecast the current data transfer speed. The same authors [14] proposed co-allocation architecture for grid data transfers across multiple connections. They provided brute-force, history-based and other techniques. Brute force works by dividing the file size equally among available flows. It does not address the bandwidth differences among the various client-server links. Past history-based co-allocation schemes not exhibit consistent performance, since performance (speed) of the each processor varies over time [13]. Many algorithms and schemes found in the literature [3][7][14][15] make the decision based on the past history, heuristics, performance in the first allocation, etc. But, in practical, server and network performance cannot be forecasted in full guarantee. Hence, several researchers proposed dynamic strategies [14][3][13].

The dynamic co-allocation mechanisms of [14] were noteworthy. In the conservative load balancing, the dataset requested is divided into disjoint blocks of equal size. Each available server is assigned one block to deliver in parallel. Once the server finishes delivering the block, another block is requested and so on, till the

entire file is downloaded. There exists a shortcoming of the faster servers that must wait for the slower server to deliver the final block. Another strategy by the same authors is the aggressive load balancing, which progressively increase the amount of data requested from faster servers and reduce the amount of data requested from slower servers or stop requesting data altogether. These schemes are calculating bandwidth at the time of delivering the block and thereby allocating the next block under the motive of utilizing faster servers. But, the idle time of faster servers awaiting the slowest server to deliver the last block is one of factor, which affects total efficiency. The other technique, Recursive Adjustment co-allocation [3] was designed under the objective of reducing the waiting time.

In all these techniques, the data set is divided into block size according to each server's bandwidth and the co-allocator assigns the blocks to each server for transferring. But, after assigning the block size, there is no guarantee that the bandwidth of the server remains constant. In other words, this technique does not cope up with the highly dynamic performance behavior of the multiple servers and their networks. Moreover, when any one of the servers becomes idle or link is broken, the alternative is not suggested in none of the existing methods.

Our proposed scheme takes care of all these factors. It neither uses predictions nor heuristics, instead dynamically co-allocate with duplication assignments and coping up nicely with the changing speed performance of the servers. The idea of duplication in allocation has already been used in multiprocessor scheduling in the yester years and applied in computational grid environments in recent years [13]. But, there are quite number of variations between computational grid and data grid, where the former one is constrained by precedence constraints, task partitioning, dependency, interprocess communication, resource reservation & allocation, process control, etc. The next section describes our strategy which alleviates the problems mentioned afore.

4 Dynamic Co-allocation Scheme with Duplicate Assignments (DCDA)

Let D be the dataset, k be the number of blocks of a dataset of fixed block size and m be the available number of servers having replicated data content. First, D is divided into k disjoint blocks (B_i) of equal size and each one of the available server is assigned to deliver, in parallel, in other words, $D = \{B_1, B_2, \ldots, B_k\}$. The strategy for partitioning the dataset into data blocks is analyzed in section 5. When the requested block is received from a server, one of the yet unassigned blocks is assigned to the server. Co-allocator repeats this process until all the blocks of a dataset are assigned. Hence, good performance servers transfer more blocks than the slow servers, which take more time deliver. In this case, the block assigned to that server is again assigned to the faster server; In other words, there may be duplication of blocks among the servers and thereby reducing the waiting time from the slower servers. Moreover, when the server becomes idle or the link to the server is broken, the entire data process is safeguarded

from disruption. A data structure of circular queue is maintained here, with k (number of blocks) as the size of the queue. The complete scheme is presented here:

---

coalloc(m, k, D)

1. [Initialization]
   1a) Partition the dataset D into k equal sized blocks B_j, j = [1..k].
   1b) All the blocks are numbered and placed in a circular queue $CQ(k)$.
   1c) CQ pointer p is initialized with 1 so as to point to its first element.
2. [Initial allocation of blocks on to the servers]
   for (i = 1 to m)
   2a) fetch block B_p from CQ and assign to server S_i
   2b) $p = (p + 1) \bmod k$
3. When a block B_j (any j, $1 \leq j \leq k$) is delivered by the server S_l (any l, $1 \leq l \leq m$),
   3a) remove block B_j from CQ, $k = k - 1$
   3b) signal the servers to stop processing of block B_j
   3c) fetch block B_p from CQ and assign to server S_l
   If CQ is empty, Go to step 6.
   3d) $p = (p + 1) \bmod k$
4. When a server S_l ($1 \leq l \leq m$) is signalled,
   4a) fetch block B_p from CQ and assign to server S_l
   If CQ is empty, Go to Step 6.
   4b) $p = (p + 1) \bmod k$
5. [Waiting for delivery or free signal from servers]
   Go to Step 3
6. [At the completion of transfer of dataset]
   When the CQ is empty, kill all the assigned data transfer processes in other servers.

For the purpose of explaining the scheme, initially, let us consider the constant rate of transfer and the data set is divided equally into k disjoint blocks. The scheme is illustrated as follows: Let us consider the example of data set of size 100 MB replicated in five servers (s1 to s5) and hence the block size is 20 MB. Also assume that the speeds of the data transfer of the five servers are 200, 70, 150, 80, 200 Kbps, respectively. Note that, when same blocks are received by co-allocator, it considers the one with the earliest timestamp and discards other.

Note that in this first example, $k = m$, for simplicity. Initially, a block numbered 1 to 5 is assigned to each server, respectively in the same order. Naturally, servers 1 and 5 delivered data much faster than others. After the blocks 1 and 5 are received, blocks 2 and 3 are assigned to servers 1 and 5. In due course, block 3 is delivered by server 3 and hence the block request to the server 5 is cancelled and block 4 is assigned to server 3 and block 2 is assigned to server 5. Note that the block 2 is duplicated in servers 1, 2 and 5. Finally, block 4 is assigned duplicated in all the sites. This is illustrated in the Gantt chart (Fig. 3a). The total time taken is 273.0 seconds. The allocation of

blocks on to the servers, for every reception of delivery signal in succession is shown as in Fig. 4a along with the status of the queue. At the end of the execution, the blocks delivered by the servers are shown in Fig. 4b.

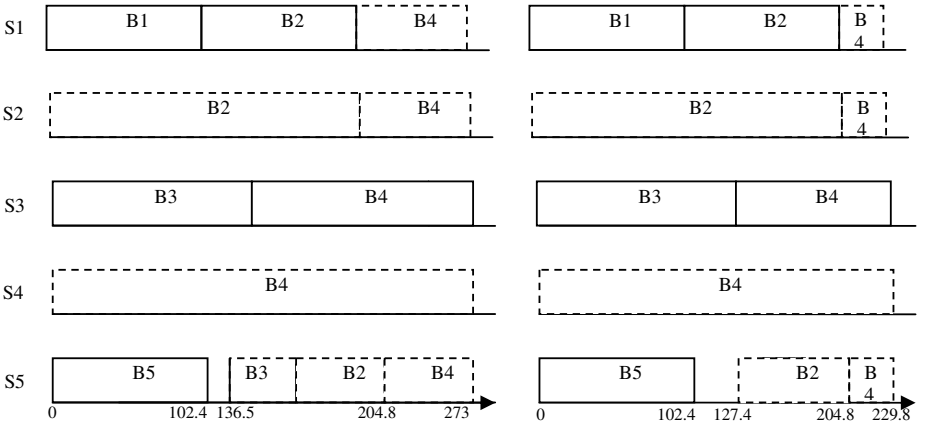


Fig. 3. Gantt Chart showing Co-allocated Behavior with a)Constant and b) varying bandwidth

The allocation of blocks on to the servers, for every reception of delivery signal in succession is shown below (Fig. 4a) along with the status of the queue. At the end of the execution, the blocks delivered by the servers are shown in Fig. 4b.

S_1	S_2	S_3	S_4	S_5
B_1	B_2	B_3	B_4	B_5
B_2	B_2	B_3	B_4	B_3
B_2	B_2	B_4	B_4	B_2
B_4	B_4	B_4	B_4	B_4

(a)

$B_1\ B_2\ B_3\ B_4\ B_5$ $CQ(5)$
$B_2\ B_3\ B_4$ $CQ(3)$
$B_2\ B_4$ $CQ(2)$
B_4 $CQ(1)$

B_1	B_2	B_3	B_4	B_5
S_1	S_1	S_3	S_1	S_5

(b)

Fig. 4. Status of the circular queue during execution and the final allocation

5 Experiment and Analysis

5.1 General Analysis

In order to analyze the scheme, let us consider the next case (case 2) of change in network performance. Let the data transfer rate, be represented as a pair (a, b) where, a is the rate of transfer in the first 100 seconds and b represents after first 100 seconds. In this way, rate of transfer for five servers are considered as (200,200), (70,50),(150,200),(80,0), and (200,150) respectively. Rate of transfer 0 specifies nil or no data transfer, which may be due to the broken link or idle server (here, site 4 detoriate, after 100 seconds). In this case, the total time taken will be 229.8. Note that the performance can further be improved, when the frequency of change-in-transfer

rate is more. Now, consider the same case with more number of blocks. In the same example cited above, partition the data set into ten blocks, each of size 10 MB. The total completion time is drastically reduced to 204.8 and the blocks delivered by the appropriate servers are tabulated as Table 1.

Table 1. Performance of Dynamic Duplicate Assignment Scheme with $k > m$

Sites	Rate of transfer (interval of 100 secs)	Blocks assigned	Blocks delivered
1	200, 200	1, 6, 9, 4	1, 6, 9, 4
2	50, 70	2, 4	-
3	150, 200	3, 8, 2, 10	3, 8, 2
4	70, 0	4	---
5	200, 150	5, 7, 10, 4	5, 7, 10

The performances of the three cases mentioned above are shown as bar chart in the Fig. 5. Numbers of blocks considered are mentioned along with the case. Thus, from the figure, it is apparent that the varying speed performances of the replicated sites are utilized which results in minimal completion time of the download process of a data set.

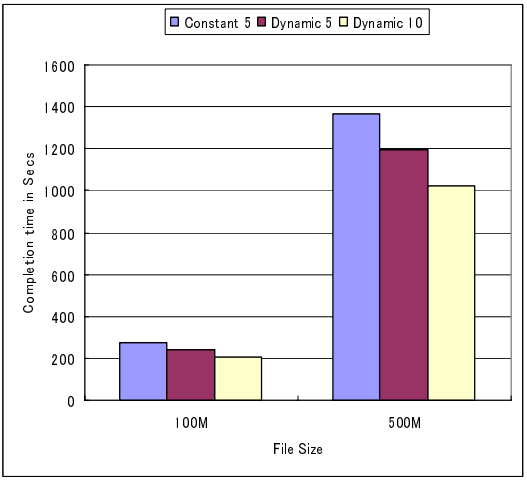
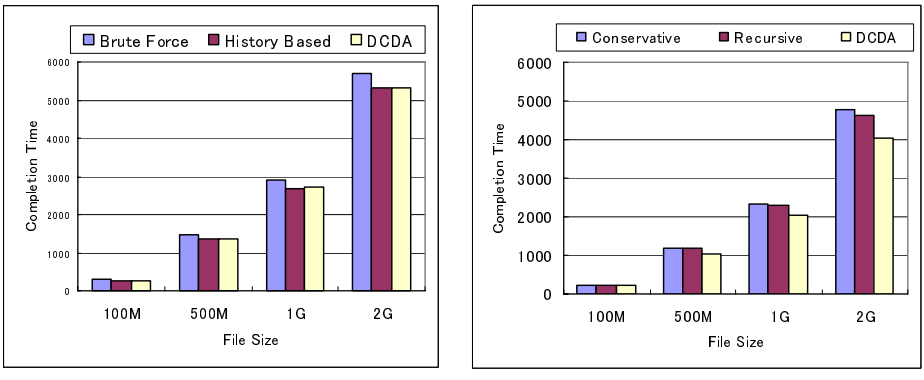


Fig. 5. Performances of Static and Dynamic bandwidth for 100M & 500M file sizes

We used Globus toolkit 2.4 in our experiment, and in order to study the performance of our scheme in varying network performance, we conducted our experiment by changing the network and server loads, apart from normal traffic. In order to evaluate our scheme in dynamic environment, we used frequency table of data transfer rate. For example, Table 2 shows one such frequency table when the case 2 above is extended with more frequency of data transfer in the interval of 20 seconds.

We can study the behavior of our algorithm with frequency tables like this. We used several file sizes as 2GB, 1.5 GB, 1GB, 500MB, 100MB and 10MB and the expected completion time threshold is assumed as 5 minutes, which is required by the 3 schemes mentioned above. We fixed a constant L=5 in the formula of finding *k*.



a) with Static schemes b) with Dynamic schemes

Fig. 6. Comparative Performance with $k > m$

We assumed that the overhead latency in assigning, delivering and killing of duplicate assignments are negligible since, in practice, while transferring giga, peta or tera bytes of sizes, these delays will not affect the overall completion time. We have evaluated our scheme with other static and dynamic schemes, separately. When the bandwidth is assumed static, it has been compared with brute force[14] and our proposed dynamic scheme is compared with other dynamic strategies of conservative load balancing scheme[14] and recursive co-allocation[3]. We analyzed the performance of each scheme by comparing their completed transfer time, shown in Fig. 6. When comparing the static schemes, we assumed constant rate of transfer (as in case 1), as in Fig. 6a. From this figure it is clear that our scheme has marginal improvement over others. In comparing with other dynamic schemes, Fig. 6b, our scheme outperforms others. Furthermore, the other schemes are not fault tolerant, and the expected completion time is specified here, for the purpose of comparison.

The blocks and the servers delivered by them in the order of arrival (left to right) for the case 2 are :

B_1	B_5	B_3	B_6	B_7	B_8	B_9	B_{10}	B_4	B_2
S_1	S_5	S_3	S_1	S_7	S_3	S_1	S_3	S_1	S_3

Fig. 7. Final Allocation of Blocks on to servers

One of the factors, which influence the scheme, is data set portioning; that is the manner in which the data set is partitioned. The question now is whether to have small number of blocks with greater size or more number of blocks with smaller size. The later one is better since more number of blocks may brighten the scope of

dynamicity; in other words, there is a possibility of assigning more blocks to the faster servers. But, at the same time, more number of blocks may have the overhead of communication latency and the block management. Next subsection discusses about fixing the block size *k*.

5.2 Finding the Optimal Number of Blocks

One of the factors, which influence the overall performance, is data set portioning; that is the manner in which the data set is partitioned. The question now is whether to have small number of blocks with greater size or more number of blocks with smaller size. The later one is better since more number of blocks may brighten the scope of dynamicity; in other words, there is a possibility of assigning more blocks to the faster servers. But, at the same time, more number of blocks may have the overhead of communication latency and the block management. The partitioning factor in turn based on block size and the number of replicated sites available. Choosing the optimal block may yield significant performance with our scheme. In general, smaller number of blocks may yield poor completion time and on the other hand, more number of blocks results in switchover overheads and thereby showing poor completion time.

Hence, it is highly important to partition the data set into optimal number of blocks. The specialty of the algorithm is independent of any estimating measures under the motive of adapting to the natural dynamicity of network behavior. Without compromising this objective let us fix the number of blocks, based on the function of bandwidth. Before executing the algorithm, assume that the bandwidths of all the servers are known from the client perspective. These metrics can easily be obtained from the monitoring module, which is executing periodically by using the tools such as *iperf* [12].

Let the ratio of coefficient of variation of set of bandwidths be, $C_v = (\sigma / \mu)*100$, where, σ is the standard deviation and μ is the average of the bandwidths from client to all the sites having replicated data. Further, the set of bandwidths in a network of multisender scenario aggregates normal distribution [10]. Hence, if set of bandwidth values aggregates normal distribution, C_v can be used to compare the amount of variance between populations with different means.Based on the basic statistics, it can be interpreted that, the lower percentage is closer to the average and the higher percentage depicts the farther distance from average. The number of fixed sized blocks can be fixed as, $k = m *([C_v / [100/L]] + 1)$, (or can be simplified as $k = m ([\sigma L/ \mu] + 1)$) for any constant L (> 0) which is used to divide the range of distributions. For example, for normal distribution curve with μ=140 and σ=2.007638, the entire range of distribution is divided in to 4(=L) portions. Note that, for this example, $k = 2m$.

5.3 Improvement of the Algorithm

Note that the sequentiality is not maintained in this method. In other words, the blocks are not received in the partitioning order. For example, the blocks and the servers delivered by them in the order of arrival (left to right) for the case 2 as in Fig.7 is not block sequential. This may not be the problem with the applications considering insensitive with sequentiality. On the other hand, for the applications like streaming, the sequential delivery of partitioned blocks is matters a lot. Hence the algorithm is

modified to ensure the sequential delivery to the client and at the same time exploiting the parallel feature enriched in the co-allocated model.

The data file is divided into blocks and each block is further partitioned into sub-blocks to exploit the parallelism in downloading. This is illustrated in the Fig. 8. The dashed lines indicate the sub-blocks.

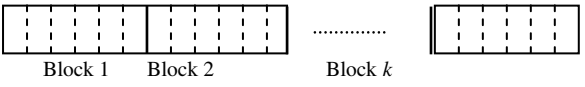


Fig. 9. Partitioning of a Data file

For each block, the basic algorithm in section 4 is executed as, *coalloc*(*m, s,* B_i), where *s* is the number of sub-blocks and B_i is the block *i* of the dataset *D*, for any *i*, $1 \leq i \leq k$. After all the sub-blocks of a block are delivered, rearranged the sub-blocks, execute for the next block *coalloc*(*m, s,* B_{i+1}) and this process is repeated for all the blocks of a dataset. Hence, the blocks will be received in the sequential order.

Table 2. Sample Frequency Table of Data Transfer Rate

	Rate of transfer in K/Second in the interval 20 seconds												
S1	200	200	80	190	210	220	205	180	150	120	180	185	200
S2	50	50	80	80	90	70	70	80	80	90	70	80	80
S3	150	160	50	150	170	180	200	210	210	190	200	210	200
S4	70	80	70	60	40	60	50	0	0	0	0	0	0
S5	200	210	190	200	200	190	170	160	150	160	150	140	150

For example, consider the simple example of case 2, discussed in section 5.1. Let the number of blocks as 4 and each block has 5 sub-blocks. Thus, the size of sub-block is 5MB. Only 4 servers S_1, S_2, S_3 & S_5 are considered, eliminating server S_4. For the purpose of explanation, we denote the sub-block with double index, as SB_{ij}, where *j* refers to the sub-block number within a block *i*. Now, with the improved sequential algorithm, the sub-blocks will be delivered in the following order, (from left to right). $SB_{11},SB_{15},SB_{13},SB_{12},SB_{14},SB_{21},SB_{25},SB_{23},SB_{22},SB_{24}...SB_{41},SB_{45},SB_{43},SB_{42},SB_{44}$

Note that there is a necessity of rearrangement of sub-blocks, before the next iteration of a block. This scheme ensures the ordered delivery of data file and thus highly suitable for the applications like streaming.

6 Conclusion

We have designed a dynamic co-allocation model, to enable parallel download of replicated data from multiple servers. The coallocation scheme is presented which initially fix the number of data blocks based on bandwidth obtained from monitor. Our scheme uses neither past history nor heuristics but fully compliant with high dynamicity in the network / server performance. The scheme works fine, even when the link to servers is broken (or servers become idle) during the process, whereas,

none of the existing algorithms considered this situation. It is compared with the existing schemes and shows significant improvement in overall completion time of data transfer. The scheme may yield significant performance when choosing optimal block size.

Acknowledgment

This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for JSPS Fellows 1604285, Scientific Research on Priority Areas 18049038 and Scientific Research (C) 18500050.

References

1. Allcock B, Bester J, et al, “Data Management and Transfer in High Performance Computational Grid Environments”, Parallel Computing, May 2002.
2. Bhuvaneswaran R.S, Katayama Y, Takahashi N ,“Dynamic Co-allocation Scheme for Parallel Data Transfer in Grid Environment”, Semantics, Knowledge and Grid, Beijing, pp 178-188, 2005.
3. Chao-Tung Yang, I Hsien Yang, Chun Hsiang Chen, “Improve Dynamic Adjustment Mechanism in Co-allocation data Grid Environments”, Proceedings of the 11th Workshop on Compiler Techniques for High-Performance Computing (CTHPC 05), 189-194, 2005
4. Chervenak A, et al ,”A Framework for Constructing Scalable Replica Location Services”, Proceedings of Super Computing Conference 2002, Baltimore, 2002.
5. Chervenak A, Foster I, et al, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets,” Journal of Network and Computer Applications, 23:187-200, 2001.
6. Chun Hsiang Chen, Chao-Tung Yang, Chuan-Lin Lai, “Towards an Efficient Replica selection for Data Grid”, Workshop on Grid Technologies and Applications, Dec 2004.
7. Czakowski K, Foster I, Kesselman C, “Resource Co-allocation in Computational Grids”, Proc. IEEE International Symposium on High Performance Distributed Computing 1999.
8. Data Grid Project (EU Data Grid), http://www.eu-datagrid.org
9. GridPhyN project (Grid Physics Network), http://www.griphyn.org
10. Hui,S.C and Jack Y. B. Lee, "Modeling of Aggregate Available Bandwidth in Many-to-One Data Transfer," Proc. of the Fourth International Conference on Intelligent Multimedia Computing and Networking, July 21-26, 2005, Utah.
11. Introduction to Grids and the Globus Toolkit, The Globus Project, http://www.globus.org.
12. Iperf Homepage : http://dast.nlanr.net/Projects/Iperf/ .
13. Noriyuki Fujimoto, Kenichi Hagihara, “Near Optimal Dynamic Task Scheduling of Independent Coarse Grained Tasks onto a Computational Grid”, International Conference on Parallel Processing (ICPP-03), pp.391-398, October 6-9, 2003.
14. Vazhkudai S, “Enabling the Co-allocation of Grid Data Transfers”, International Workshop on Grid Computing, Nov 2003, pp 44-51.
15. Vazhkudai S, Tuecke S, Foster I, “Replica Selection in the Globus Data Grid”, IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2001, pp 106-113