A Dynamic Scheduling Approach for Coordinated Wide-Area Data Transfers using GridFTP [*]

Gaurav Khanna [1], Umit Catalyurek [2], Tahsin Kurc [2], Rajkumar Kettimuthu [3],
P. Sadayappan [1], Joel Saltz [2]

[1]Department of Computer Science and Engineering, The Ohio State University
[2]Department of Biomedical Informatics, The Ohio State University
[3] Mathematics and Computer Science Division, Argonne National Laboratory

Abstract

Many scientific applications need to stage large volumes of files from one set of machines to another set of machines in a wide-area network. Efficient execution of such data transfers needs to take into account the heterogeneous nature of the environment and dynamic availability of shared resources. This paper proposes an algorithm that dynamically schedules a batch of data transfer requests with the goal of minimizing the overall transfer time. The proposed algorithm performs simultaneous transfer of chunks of files from multiple file replicas, if the replicas exist. Adaptive replica selection is employed to transfer different chunks of the same file by taking dynamically changing network bandwidths into account. We utilize GridFTP as the underlying mechanism for data transfers. The algorithm makes use of information from past GridFTP transfers to estimate network bandwidths and resource availability. The efficiency of the algorithm is evaluated on a wide-area testbed.

1 Introduction

Grid computing technologies have enabled scientists to generate, store, and share data distributed across multiple sites. Data analysis in a Grid setting involves use of distributed collections of storage and computational systems and transfer of large volumes of data in a wide-area network. An example is the LHC [1] experiment at CERN. The data which is generated by a CMS experiment at LHC needs to be transferred to a Tier-1 site in the US where it is processed and then multi-cast onto many domestic US tier-2 sites. As another example, consider a multi-institutional study which collects and analyzes biomedical image data, obtained from high-resolution scanners to develop animal models of phenotype characteristics in disease progression. Hundreds or thousands of images can be obtained from a subject and there can be hundreds of subjects in a study. These images may be collected and stored at multiple sites. Researchers wishing to carry out an analysis using images from a large population of subjects will query image datasets at multiple sites. The image files extracted as a result of the query will then either be downloaded to a local system or be transferred to computational machines distributed in the environment for processing. These scenarios involve transfer of large volumes of files from the storage sites to the computational sites. Addressing this problem requires efficient coordination of data movement across multiple source sites, destination sites, and intermediate locations over the wide-area network.

In this work, we seek efficient algorithms to schedule and execute the transfer of a set of files distributed across multiple machines to another set of machines in a wide-area environment. The objective is to minimize the total execution time of a batch of file transfer requests. A destination machine receives a subset of the files. The subsets of files assigned to different destination machines may overlap, i.e., a file may be mapped to multiple destination machines. Figure 1(a) illustrates an example of the problem. Files labeled by Fi are stored on distributed storage repositories. A subset of files are to be transferred to disks on a distributed set of compute nodes, denoted by Ni, over a wide-area network. Figure 1(b) shows that two different sources of a file $F1$ can be used simultaneously to transfer disjoint chunks of the file, thereby increasing the throughput. The figure shows that once a replica of $F1$ is created on the node

[*]This research was supported in part by the National Science Foundation under Grants #CCF-0342615, #CNS-0403342 and #CNS-0643969.

$N1'$, then the node $N1'$ and the storage repository $D3$ can simultaneously transfer the file to the node $N1$.

We present a network flow based mixed integer programming (IP) formulation of the scheduling problem. The resulting solution is a lower bound on transfer time under idealistic conditions of resource availability and performance. We then propose a dynamic scheduling heuristic which employs network bandwidth information obtained from past GridFTP transfers to adapt its scheduling decisions, thereby, accounting for the resource availability fluctuations in the wide-area environment. The algorithm also employs adaptive replica selection, if files are replicated in the environment during previous transfers. It performs simultaneous transfer of portions of files from multiple replicas to maximize data transfer bandwidth. We have developed an implementation of our algorithm using GridFTP [4] as the underlying transport protocol for data transfers. We experimentally evaluate the algorithm on a wide-area network testbed consisting of clusters located at geographically disparate locations. The results show that the algorithm can take advantage of multiple replicas and concurrent data transfers.

2 Related Work

GridFTP [4] is a widely used protocol which enables secure, reliable and high performance data movement. It facilitates efficient data transfer between end-systems by employing techniques like multiple TCP streams per transfer, striped transfers from a set of hosts to another set of hosts and partial file transfers. In this work, our contribution is a new dynamic scheduling scheme to collectively schedule a batch of file transfer requests. Our approach performs adaptive replica selection and simultaneous transfer of a file from multiple replicas. In this paper, we have applied our approach in conjunction with GridFTP, that is, the scheduling algorithm employs GridFTP as the file transfer protocol.

BitTorrent [13] is an incentive-based file sharing system which employs a tit-for-tat strategy where in the peers which contribute more data at faster rates get preferential treatment for downloads. In BitTorrent, a source peer can upload data to upto 5 other peers simultaneously, the chosen peers being the ones which provide the highest upload rates to the source peer. In addition, it also incorporates the concept of optimistic unchoking where in periodically, a source peer chooses a randomly selected peer from the set of requesting ones and starts uploading to it. This allows a site to discover peers that possess the data of interest and that may be able to upload to it at higher rates regardless of the site's upload rate. However, the key principle on which BitTorrent relies upon is based on incentives. In this work, the goal is to minimize the total transfer time in a collaborative setting where the global objective of minimizing the time is more important than each site's local benefits. Bullet [11], Chainsaw [12] and Splitstream [5] are examples of systems designed for high-bandwidth data dissemination, motivated by applications like real-time multimedia streaming. However, all these systems target data distribution from a single source to a large number of receivers, that is, multicast. In our work, we target the scheduling problem in a very generic context which involves coordinated data movement across multiple sources, destinations and intermediate sites.

Stork [10] is a specialized scheduler for data placement activities on the Grid. The scheduler allows checkpointing and monitoring of data transfers as well as use of DAG schedulers to encapsulate dependencies between computation and data movement. In this paper, we focus on modeling the heterogeneity and the dynamics of a wide-area environment to perform efficient collective file transfer scheduling. Swany et al. [14] exploits the "logistical effect" which essentially means improving performance by dividing a connection into a series of shorter, better performing connections. Giersch et al. [6] have addressed the problem of scheduling a collection of tasks sharing files onto heterogeneous clusters. Their work focused mainly on task mapping and they proposed extensions to the MinMin heuristic [7] to lower the scheduling cost. In our past work, we looked at the problem of scheduling a batch of data-intensive tasks [9]. We have also investigated scheduling of file transfers in data center environments where in the scheduler has ultimate control [8]. In this work, we are targeting dynamic heterogeneous wide-area environments like Grids.

3 Network Flow Formulation

In this section, we propose a mixed integer programming (IP) formulation of our target problem. The formulation is based on the maximization of network flows from sources to sinks. The wide-area environment is represented by a graph $G = (V, E)$, referred to here as the *platform graph*. In this graph, V is the set of machines and E represents the network edges. A network edge is the wide-area connection between two machines. The weight of the edge is a measure of the achievable bandwidth between the two machines. The set of two tuples $R = \{< f_\ell, v_d >\}$ represents that file f_ℓ needs to be transferred to the destination node v_d. The set of two tuples $D = \{< f_\ell, v_s >\}$ denotes that file f_ℓ is present on the source node v_s. Multiple replicas of a file may exist and each replica is represented by a two

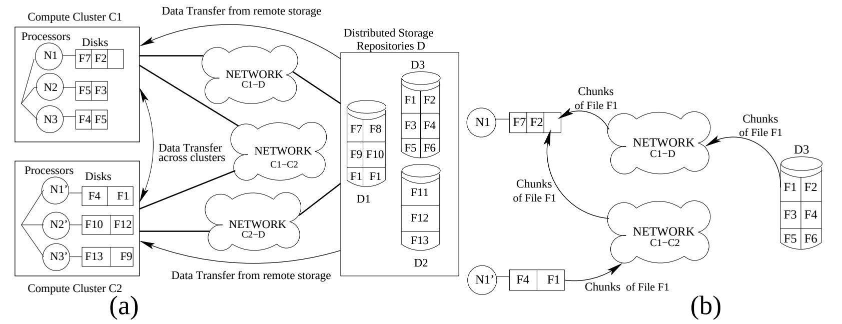


Figure 1. (a) The wide-area data staging problem, (b) Simultaneous usage of multiple replicas of File F1

tuple in D.

The optimization problem solves for a set of variables $Flow_{ij\ell}$, where $Flow_{ij\ell}$ is the rate (bandwidth) at which file f_ℓ is transferred through the link between the nodes v_i and v_j.

Let $InFlow_{i\ell}$ be the rate at which the file f_ℓ enters the node v_i along the incoming edges.

$$(\forall \ell)(\forall i, i \in V) InFlow_{i\ell} = \sum_{(\forall j,(j,i)\in E)} Flow_{ji\ell} \tag{1}$$

For each file f_ℓ, the flow on each outgoing edge which emanates from the node v_i cannot exceed the inflow at which the file f_ℓ enters the node v_i. This necessarily holds true for all the nodes except the source node set for the file f_ℓ.

$$(\forall \ell)(\forall j)(\forall i, i \in V - \{k| < f_\ell, v_k > \in D\}) \\ Flow_{ij\ell} \leq InFlow_{i\ell} \tag{2}$$

The total inflow rate of all the files entering a node v_i should not exceed the bandwidth capacity at the node v_i, $VCap(i)$. Similarly, the total outflow rate on all outgoing edges should not exceed the bandwidth capacity at the node v_i.

$$(\forall i, i \in V) \sum_{(\forall \ell)} InFlow_{i\ell} \leq VCap(i) \tag{3}$$

$$(\forall i, i \in V) \sum_{(\forall j)(\forall \ell)} Flow_{ij\ell} \leq VCap(i) \tag{4}$$

The aggregate flow rate for all the files through the edge e between the nodes v_i and v_j should not exceed the bandwidth capacity $ECap(ij)$ of the edge e.

$$(\forall i, i \in V)(\forall j, (i,j) \in E) \sum_{(\forall \ell)} Flow_{ij\ell} \leq ECap(ij) \tag{5}$$

We only consider destination nodes as intermediate nodes for other transfers. Therefore, an edge from a node v_i to a node v_j can have a non-zero flow for a file f_ℓ, only if the node v_j belongs to the destination node set for the file f_ℓ.

$$(\forall i, i \in V)(\forall j, (i,j) \in E)(\forall \ell, < f_\ell, v_j > \notin R) \\ Flow_{ij\ell} = 0 \tag{6}$$

A feasible solution should not have flow cycles for each file f_ℓ. In other words, for each file f_ℓ, for all cycles in the graph G comprising only a subset of destination nodes for the file f_ℓ, the flow for the file on atleast one of the edges belonging to the cycle should be equal to zero. Let $Cycles_\ell$ be the set of all the cycles in the graph G consisting only of a subset of destination nodes of the file f_ℓ . Each element of the set $Cycles_\ell$ is a set of edges which constitute that cycle.

$$(\forall \ell)(\forall C, C \in Cycles_\ell)(\exists (i,j), (i,j) \in C) Flow_{ij\ell} = 0 \tag{7}$$

The finish time $FinishTime_{\ell k}$ of a transfer request for a file f_ℓ to its destination v_k is computed as follows.

$$FinishTime_{\ell k} = \frac{FileSize(\ell)}{InFlow_{k\ell}} \tag{8}$$

Note that the finish time is computed based on the total incoming flow to the destination node v_k for the file f_ℓ. We cannot use the total outgoing flow from the sources of the f_ℓ to compute the finish time since there are possibly multiple destinations for each file and therefore outgoing flow from a source node for a particular file is not necessarily the aggregate flow for a particular file-destination pair.

The objective is to minimize the total transfer time $Makespan = \max_{\forall \ell, k} FinishTime_{\ell k}$. Note the objective function is a non-linear function which means that the problem is non-linear optimization problem with linear constraints. We represent the objective function in an alternate way which makes the problem a linear optimization problem. We define a function $NormalizedRate_{\ell k}$ for each file transfer request.

$$NormalizedRate_{\ell k} = \frac{InFlow_{k\ell}}{FileSize(\ell)} \quad (9)$$

With the new formulation, the objective becomes the maximization of $MinRate$, which is the minimum value of $NormalizedRate_{\ell k}$ over all file transfer requests. The solution to this optimization problem will provide, for each file transfer request, the flow rates which the request should employ for each edge in the graph. These flow rates can then be used to find the total transfer time, $Makespan$. This value of total transfer time acts as a lower bound under idealistic conditions of resource availability and performance.

The flow based IP formulation inherently assumes that all file transfers take place in parallel and simultaneous file transfers on the same link share bandwidth. From a theoretical standpoint, serializing transfers on a resource, or simultaneous execution with resource sharing, results in the same makespan. In practice, however, because of limited resources on nodes and the high cost of congestion on lossy wide-area links, which leads to a continuous loop between TCP slow-start and congestion-avoidance phases, the solution obtained by the IP cannot be achieved for large batches with thousands of requests. Moreover, the scheduling overhead of a mixed integer programming approach may be unacceptable, especially for large workloads and system configurations. Therefore, in this paper, we employ the solution obtained by the IP as a lower bound on the total transfer time and use it as a yardstick to compare against our proposed dynamic scheduling heuristics which we discuss in detail in Section 4.

4 Dynamic Scheduling Algorithms

In our approach, scheduling is done per chunk basis. Chunk is a portion of the file being staged to a destination machine. Transfer of chunks for a file can be inter-leaved with transfer of chunks for other files. Our scheduling approaches are iterative, employ adaptive replica selection, and use of multiple sources for simultaneously transferring multiple pieces of the same file, i.e., non-overlapping portions of a chunk, *sub-chunks*, can be retrieved simultaneously from multiple file replicas. In a wide-area environment, the network is often the bottleneck. A good choice of replicas along with concurrent transfer of data can be expected to yield good performance. Thus, given a graph G, the set of tuples $R = \{< f_\ell, v_d >\}$, the set of tuples $D = \{< f_\ell, v_s >\}$, our objective is then to compute a schedule that will minimize the total file transfer time. The schedule comprises of a set of four tuples $< V_s, v_d, c_\ell, t >$. Here, c_ℓ is a chunk of file f_ℓ to be transferred, v_d is the destination machine, V_s is the set of source machines, from which portions of the chunk c_ℓ will be transferred, and t is the time at which the transfer of the chunk will start.

Replica selection depends upon a number of factors like network bandwidths, round-trip times, and file sizes. Moreover, in a wide-area network, the network bandwidth may fluctuate considerably. In order to handle dynamic network characteristics, our approach carries out replica selection "at the level of chunks" in an adaptive manner. We should note that as files are staged to their respective destination nodes, these nodes can act as replica sources for other requests of the same file. We employ dynamic information obtained on the fly from previously executed file transfers to drive our scheduling and replica selection decisions.

In order to support adaptive replica selection at chunk level and concurrent use of multiple replicas, we redefine the request set R and the data structure D. We define the modified request set R' as the set of three tuples, $R' = \{< f_\ell, v_d, cur\_request\_offset(\ell, d) > \mid < f_\ell, v_d > \in R\}$ denoting that f_ℓ needs to be transferred to the node v_d starting at the offset $cur\_request\_offset(\ell, d)$. This means that a subset of the file f_ℓ is already present at the node v_d up to an offset $cur\_request\_offset(\ell, d)$. The value of $cur\_request\_offset(\ell, d)$ will change with time as more and more chunks of file f_ℓ get written onto node v_d. The initial values of $cur\_request\_offset(\ell, d)$ are set to 0, since the transfer of a file will start at offset 0. Similarly, D is redefined as $D' = \{< f_\ell, v_s, last\_byte\_offset(\ell, s) > \mid < f_\ell, v_s > \in D\}$, representing that the file f_ℓ is currently present on the node v_s up to the offset value $last\_byte\_offset(\ell, s)$. Here, v_s can be one of the original source nodes of the file, or a destination node, to which the file has already been partially transferred. In the case v_s is a destination node, the value of $last\_byte\_offset(\ell, s)$ will change over time as more and more chunks become available on v_s. The final value of $last\_byte\_offset(\ell, s)$ will be the size of the file, $size(f_\ell)$.

4.1 Global Dynamic Scheduling Algorithm

This scheduling scheme proceeds in steps and in each step it selects a pending file transfer request $< f_\ell, v_d, cur\_request\_offset(\ell, d) >$ from R' and computes a schedule for the request. A request is considered pending if the file associated with the request has not been completely transferred to its corresponding destination and no other chunk of this file is being transferred to the same destination. The schedule for a request consists of a four tuple with the following elements: (1) the set of replica locations (V_s) to be accessed to retrieve the data, (2) the size of the chunk ($ChunkSize$) which will be scheduled for transfer at the current scheduling instant, (3) the portions of the selected chunk to be obtained from each source, and (4) the TCP buffer sizes to be used for each connection.

In our current implementation, we employ GridFTP as the underlying transfer mechanism. Each source node runs a GridFTP server. Each destination node uses the GridFTP client side API to retrieve the portions of the file. Since a destination node can become a replica source for a file, a GridFTP server runs on each destination node as well. After the schedule for a chunk has been computed, the scheduler sends the schedule information to the corresponding destination node. The destination node starts the retrieval of the chunk from the source nodes. The scheduler moves on to the next pending file transfer request and repeats the whole process. The overall scheduling scheme is illustrated in Algorithm 1.

At step 7, the replica selection method denoted as $SelectReplicas$ is invoked to select replicas for the transfer request; the algorithm for replica selection is described in the next section. The output from this method makes up the schedule for the request. The next step (step 8) is to compute the expected minimum completion time for transferring a chunk of the requested file. The transfer completion time is computed as follows. We first divide the aggregate chunk of size $ChunkSize$ into sub-chunks which will be fetched from each replica. The size of the sub-chunks are chosen to be in the same ratio as that of the bottleneck bandwidths between each source host and the destination. The transfer completion time is then simply the maximum of the times taken to send each sub-chunk from a source to the destination. At step 9, following the well-known MinMin [7] algorithm, among all the pending requests, the file transfer request with the minimum expected completion time is chosen to be scheduled on the set of resources which yield its minimum completion time. The overall process repeats until all the file transfers have been scheduled. The replication selection step, the determination of the chunk size, and dynamic bandwidth prediction are presented in detail in the following sections.

Algorithm 1 Global Dynamic Scheduling Heuristic

```
Input: Platform $G = (V, E)$ and a set $R = \{< f_\ell, v_d > \mid$ file $f_\ell$ is requested by destination $v_d\}$
1:  $R' = \{< f_\ell, v_d, 0 > \mid < f_\ell, v_d > \in R\}$ { start transfer of each file from offset 0}
2:  $D' = \{< f_\ell, v_s, size(f_\ell) > \mid < f_\ell, v_s > \in D\}$
3:  $HostBw_i =$ the host bandwidth at node $v_i$
4:  while there are pending requests, i.e., $R' \neq \emptyset$ do
5:    if $\exists v_d$ such that $HostBw_d > \epsilon$ then
6:      for each request r = $< f_\ell, v_d, cur\_request\_offset(\ell, d) > \in R'$ do
7:        $< V_s, ChunkSize, TCPBufSize, SubChunkSize > \leftarrow SelectReplicas(G, D', r)$
8:        Compute the expected finish time to transfer the chunk of file $f_\ell$ to destination $v_d$.
9:      Choose the request $r$ with the minimum expected finish time
10:     Schedule the transfer of the chunk of the file $f_\ell$ from replica nodes $V_s$ to the node $v_d$.
11:     $R' \leftarrow R' - \{r\}$
12:     Update the expected available host bandwidth ($HostBw_i$) at the source and destination nodes.
13:   for every completed chunk transfer $< V_s, v_d, c_\ell, t >$ do
14:     Update the available network bandwidths between sources ($v_s \in V_s$) and node ($v_d$)
15:     if $endOffset(c_\ell) < size(f_\ell)$ then
16:       $R' \leftarrow R' \bigcup \{< f_\ell, v_d, last\_byte\_offset(endOffset(c_\ell), d) > \}$
```

4.1.1 Replica Selection

The replica selection algorithm (Algorithm 2) proceeds as follows. For each replica location v_s, we record the bandwidth obtained through past GridFTP transfers to find the network bandwidths and end-to-end latencies from the location v_s to the destination v_d. To perform replica selection, we apply a two phase heuristic. Each phase involves applying a filtering condition to choose a subset of replica sources of the file to fetch the data. The first filtering condition is based on the file size and its relation to the slow start phase of TCP. The second filtering condition is based on the expected available bandwidth at the sources and destination of files as well as the expected available bandwidth in the network. The output of the second filtering condition is a subset of replicas to be used for transferring the file. The TCP buffer size and the size of the portion of the chunk to be fetched from each replica are also computed.

Algorithm 2 Replica Selection Algorithm

```
Input: A pending request $< f_\ell, v_d, off >$
1:  for Each existing replica $v_s$ of the file $f_\ell$ do
2:      Compute the bandwidth delay product $BDP_s = NetBw_{s,d} \times RTT_{s,d}$ for the link between hosts $v_s$ and $v_d$.
3:      if $size(f_\ell) \geq C \times BDP_s$ then
4:          add the replica $v_s$ to the tentative replica set $T_s$
5:  for each replica $v_s \in T_s$ in decreasing order of available bandwidth values to $v_d$ do
6:      Add the replica $v_s$ to the final replica set $V_s$
7:      Update the destination $HostBw_d$ to account for the transfer between $v_s$ and $v_d$ (if $NetBW_{s,d} > HostBw_d$ the transfer bandwidth between $v_s$ and $v_d$ will be $HostBw_d$)
8:      if $HostBw_d < \epsilon$ then
9:          break
10: if $V_s = \emptyset$ then
11:     pick the source $v_s \in T_s$ with highest bandwidth and set $V_s \leftarrow \{v_s\}$
12: Compute $ChunkSize$, $TCPBufSize$, $SubChunkSize$ per replica, using $V_s$ and $v_d$ and available network bandwidth
13: return $< V_s, ChunkSize, TCPBufSize, SubChunkSize >$
```

TCP is a window-controlled transport protocol and the performance of a TCP connection is dependent on the Bandwidth-Delay product(BDP). The BDP of a network path is defined as the product of bandwidth Bw of the path and the round-trip time RTT. TCP has an initial slow start phase where in it gradually increases the send window size. If the TCP buffer size equals the BDP, the connection will be able to saturate the path, achieving the maximum possible throughput. However, if the amount of data to be transferred is lower than the BDP, the observed bandwidth will be smaller than the maximum achievable bandwidth. Hence, if the file size is smaller than a pre-determined multiple of BDP (step 3 in the algorithm), the replica is tentatively not considered for selection. Otherwise, the replica is added to the list of tentatively selected replicas T_s. The output of this phase yields a subset of replicas.

In the second phase, the subset of replicas, T_s, is further pruned based on the network bandwidth between each replica and the destination host and the bandwidth available at the destination hosts. Employing too many replica sources in parallel may overwhelm the destination host in which case each TCP connection may lose packets and hurt performance. Therefore, the replica sources should be chosen in a manner so that the aggregate in flow rate of packets matches the available bandwidth at the destination host. We use a greedy algorithm for selecting sources. For each replica location, a bottleneck bandwidth is computed as the minimum of the expected network bandwidth and the available bandwidth at the destination. We order the replica sources of the selected subset of replicas in non-increasing order of available bandwidth values to the destination node v_d, and choose them one by one until we saturate the bandwidth of the destination host.

If no replica is selected at the end of this phase, the best replica is chosen for the file and added to the set V_s (step 11). The best replica is simply the replica which yields the least completion time for the transfer and is chosen by taking into account the bandwidths from each replica location.

4.1.2 Chunk Size

The size of a chunk is decided statically. For a file transfer request, it is the maximum of a pre-determined fraction of the file size and a threshold value. The motivation behind this is the slow start and congestion control mechanism of TCP. If the size of the chunk on a certain network edge is less than the BDP, the transfer of the chunk will finish in the slow-start phase, thereby not permitting use of the maximum achievable bandwidth. The threshold value for a given file transfer request is computed as a pre-determined multiple of the sum of the BDPs between each source replica and the destination node.

4.1.3 Dynamic Bandwidth Prediction

The bandwidth to access data from a file replica is an important factor in replica selection. Replicas with higher access bandwidth are expected to give better performance. The key issue is to determine an accurate measure of expected bandwidth from a replica. We employ bandwidth information obtained from previous GridFTP transfers to predict the future access bandwidths. For each file transfer that has finished so far, we track and save the information about the achieved bandwidth between the source-destination pair into a circular queue. We employ simple mean-based predictors to estimate the value of the bandwidth in the next interval. In future, we plan to employ more sophisticated techniques [16] for more accurate bandwidth predictions.

In addition, we employ a dynamic bandwidth scaling mechanism which works in a control feedback loop as follows. If the observed bandwidth between a given source-destination pair is able to meet a certain percentage of the expected bandwidth value for N successive transfers using the source-destination pair, the expected network bandwidth for the next file transfer between the two nodes is scaled up by a pre-determined constant, BW_SCALE. The new value of expected bandwidth is then used to calculate the TCP buffer size for the file transfer between the two nodes. However, If the observed bandwidth between a given source-destination pair is lower than a certain fraction of the expected bandwidth value for N successive transfers that use the source-destination pair, the expected network bandwidth for the next file transfer between those two nodes is scaled down by BW_SCALE.

4.2 Local Dynamic Scheduling Algorithm

The global dynamic scheduler presented in Section 4.1 coordinates all the data-transfers between multiple sources and destinations. In this section, we describe a simplified variant of the global dynamic scheduler, which only uses local information in each destination node. The key idea here is that clients act independently and there is no master which coordinates multi-site file transfers. Each client (destination node) makes requests for files it needs one by one irrespective of what other clients are doing. For each file transfer, a client employs dynamic bandwidth information obtained from past file transfers and uses multiple replicas to optimize the transfer time of each file transfer. In other words, the local scheduler employs optimizations to minimize the transfer time of each file in much the same way as the global dynamic scheduler. The difference is that the scheduling decisions is made by each destination node independently. The scheduling strategy is illustrated in Algorithm 3.

Algorithm 3 Local Dynamic Scheduling Heuristic

```
Input: Platform $G = (V, E)$ and a set $R = \{< f_\ell, v_d > \mid$ file $f_\ell$ is requested by destination $v_d\}$
On each destination node $v_d$ independently do
for each file request $< f_\ell, v_d >$ in non-decreasing file size order do
    for each chunk of file $f_\ell$ do
        $< V_s, ChunkSize, TCPBufSize, SubChunkSize > \leftarrow SelectReplicas(G, D', r)$
        Schedule concurrent transfer of the chunk of file $f_\ell$ from replica nodes $V_s$ to node $v_d$.
        When transfer completes, update the available bandwidths between sources ($v_s \in V_s$) and the destination node ($v_d$)
```

	BMI	CSE	ORNL	ANL
BMI	880	880	100	4
CSE	880	880	120	4
ORNL	100	120	900	10
ANL	4	4	300	700

Table 1. Link bandwidths (Mbps) between a pair of nodes located at different sites.

5 Experimental Results

We compare our dynamic scheduling approaches against the optimistic lower bounds we obtained via IP formulation and a baseline strategy, referred to here as `Naive Scheduling`. In the baseline strategy, each destination node picks a randomly chosen replica source for retrieving a file instead of employing dynamic bandwidth information or multiple replicas.

5.1 Experimental Setup

We employ GridFTP [4] as the file transfer protocol. GridFTP exposes a set of API calls [2] for setting the TCP buffer sizes and for obtaining portions of a file from a source. In our implementation, a master scheduler sends control information to clients (destination hosts). Each destination host calls $globus\_ftp\_client\_partial\_get()$ to inform a source of the file it needs along with the start and end offsets. This is followed by a series of asynchronous $globus\_ftp\_client\_register\_read()$ calls which are used to transfer data from the source.

The experiments were carried out across 4 clusters that are located at geographically distributed sites. The first site, the BMI cluster, is a memory/storage cluster at the Department of Biomedical Informatics at the Ohio State University. The cluster consists of 64 nodes with an aggregate 0.5 TBytes of physical memory and 48TB of disk storage. The second site, the CSE cluster, is a 64 node cluster located at the Department of Computer Science and Engineering at the Ohio State University. Each node of the cluster is equipped with two 3.6 GHz Intel processors and 2 GBytes main memory. The other two sites belong to the Teragrid [15] network. One of them is the ORNL NSTG cluster which consists of 28 dual processor 3.06 GHz Intel Xeon nodes. The other one is the UC/ANL IA-32 Linux cluster which consists of 96 dual-processor Intel Xeon nodes. Table 1 shows the bandwidths in Mbps(Megabits per second) between pair of nodes from different sites.

For evaluation, we compared the performance of the various scheduling schemes under a varying set of scenarios covering different file replica distributions, file-to-destination mappings and chunk sizes. For the experimental workloads, we employed three different file sizes corresponding to the files to be transferred. The sizes were 10MB, 50MB and 500MB respectively. In each workload, the fraction of the total number of files to be transferred for each file size was decided based on the distribution of these three file sizes in the GridFTP traces obtained from Globus metrics for a recent 12-month period [3]. The fraction of the number of files of each type is 0.5, 0.35 and 0.15 respectively for the 10MB, 50MB and 500MB files.

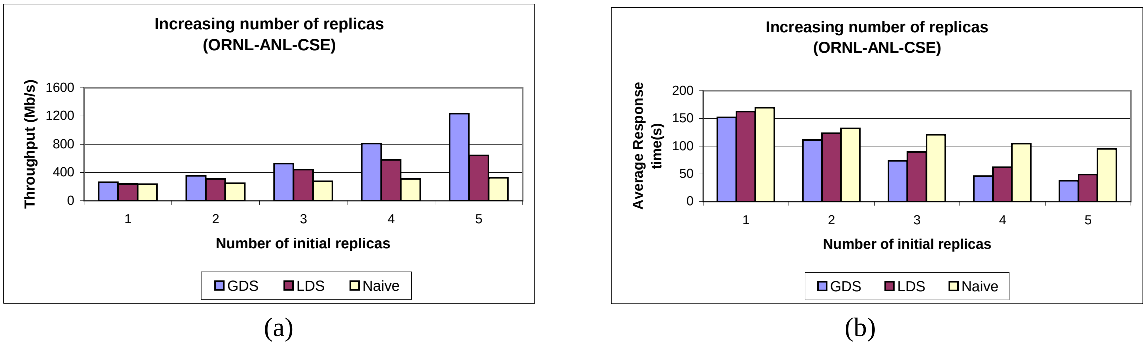
We measure the performance in terms of two metrics, namely, the average throughput which is the ratio of the total data transferred to the total execution time, and the average response time over all the requests in the workload.

5.2 Performance Evaluation

Figure 2 shows the relative performance of the Global Dynamic Scheduling (`GDS`), Local Dynamic Scheduling (`LDS`) and `Naive Scheduling` schemes on workloads with increasing degree of replication. This experiment was conducted across the 4 sites (BMI-ORNL-ANL-CSE) in a (4-3-2-3) configuration. The numbers in the parentheses refer to the number of nodes employed at each site, respectively. The input request set consisted of 300 files, the size of each of which is one of the three aforementioned values. In addition, the request set consisted of multiple destination node mappings for each file. In this experiment, all the requests in the input set had their destination as one of the nodes of the BMI cluster. The degree of replication here refers to the average number of file replicas present in the environment. Initially, the replicas were placed only on the ORNL nodes (average number of initial replicas being 1 or 2). Then, the degree of replication is increased by placing more file replicas on the ANL and CSE nodes. For the cases where the average number of replicas is one or two, all the file transfers employ either the ORNL cluster (initial replicas) or the BMI cluster (as files are created on the BMI nodes, they themselves can act as file replicas). The node-to-node bandwidth from an ORNL node to a BMI node is around 100Mbps. However, the bandwidth for a send from an CSE node to a BMI node is around 880Mbps. Therefore, as the degree of replication increases, the average throughput shows a significant performance improvement for the `GDS` scheduler. This is because, as replicas are placed on the CSE cluster, the algorithm makes an intelligent choice of choosing the CSE replicas more often than the other replicas. The results also show that the `GDS` is able to consistently outperform the other two approaches. In the other two schemes, clients act independently and make requests for files without any coordination. Each file needs to be sent to multiple different destinations, leading to increased end-point contention due to multiple simultaneous requests for the same file. Therefore, the performance improvement in these schemes due to increased replication is offset by the endpoint contention caused due to uncoordinated local scheduling. In terms of the average response time, `GDS` performs the best. `GDS` schedules the requests with the minimum expected completion time first. On the other hand, in `LDS` and `Naive Scheduling`, since multiple clients act independently of each other, requests with higher expected completion times can possibly execute before requests with lower expected completion times, thus increasing the overall response time.

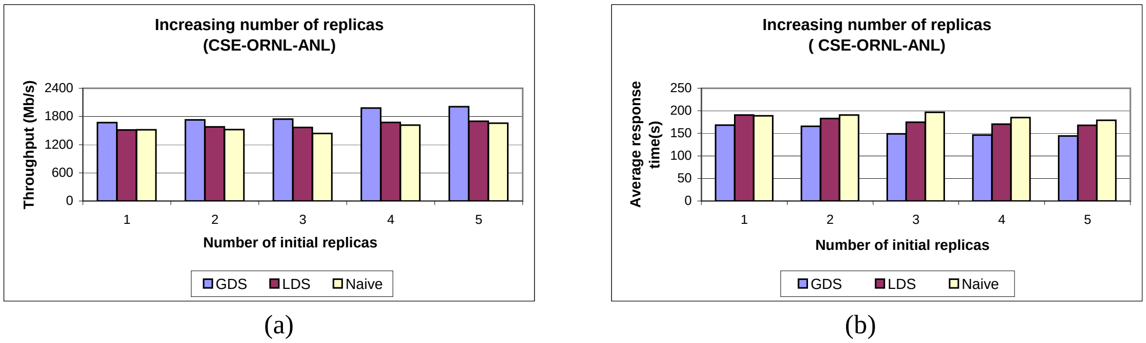
Figure 3 shows the relative performance of the various scheduling schemes with increasing degree of replication. However, in this case, the initial replication is handled differently. The replicas were initially placed only on the CSE nodes. The degree of replication is then increased by placing more file replicas on the ORNL and ANL nodes respectively. This experiment was conducted by employing the same system configuration employed in the experiment corresponding to Figure 2. The input request set consisted of 1500 file transfers. The results show that as the number of replicas increase, the average throughput does not show a significant increase, as expected. More and more replicas were placed on nodes with low link bandwidths to the destination, resulting in no significant performance improvement.

Figure 4(a) shows the the relative performance of the various scheduling schemes on workloads with increasing number of clients (destination hosts). This experiment was conducted across the 4 sites (BMI-ORNL-CSE-ANL) in a (10-3-3-2) configuration. The numbers in the parentheses are the number of nodes employed at BMI, ORNL, CSE, and ANL, respectively. During the experiment, the number of nodes on the BMI cluster was varied from 4 to 10. Each request in the input set was destined to one of the BMI nodes. The number of requests in the input set varied from 300 for the 4 BMI nodes to around 600 file transfers for the 10 BMI nodes. Again, the request set consisted of multiple destination node mappings for each file. The degree of replication in these experiments refers to the average number of file replicas initially present. The average number of initial file replicas was set to 5. The figure shows that as the number of clients increase, the throughput increases. This is because, as file replicas are created on BMI nodes, these replicas also act as sources for other



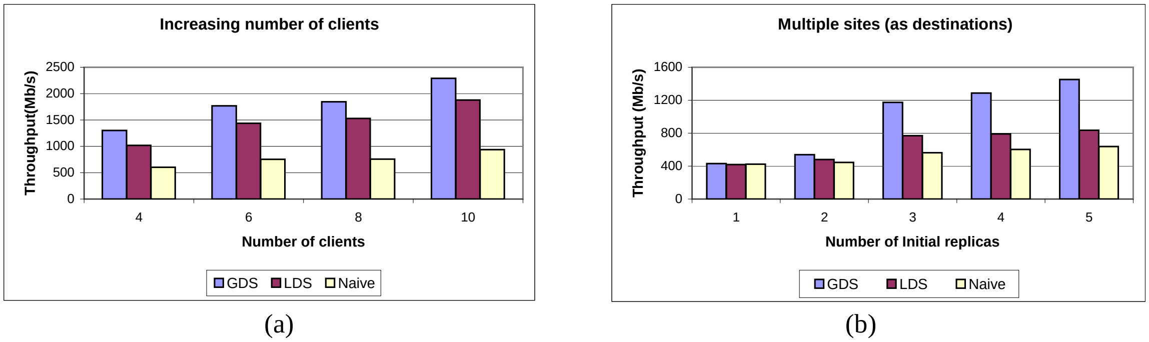
(a) (b)

Figure 2. Performance of all the algorithms with increasing number of replicas (replicas added in the order ORNL-ANL-CSE) in terms of the (a) Average throughput and (b) Average response time.



(a) (b)

Figure 3. Performance of all the algorithms with increasing number of replicas (replicas added in the order CSE-ORNL-ANL) in terms of the (a) Average throughput and (b) Average response time.



(a) (b)

Figure 4. (a) Performance in terms of throughput (Mbps) of all the algorithms with increasing number of clients (b) Performance in terms of throughput (Mbps) of all the algorithms for workloads where multiple sites act as clients as well as sources.

requested transfers of the same file. Even though the aggregate amount of transferred data increases as the number of BMI clients increases, the aggregate bandwidth increases by a greater factor, leading to increased throughput. Furthermore, the extent of performance improvement is maximum for the `GDS` scheduler. As the number of clients increase, the effects of end-point contention is expectedly higher. `GDS` makes a better job of accounting for contention by making efficient coordinated scheduling decisions, whereas the other schemes make client-side local decisions which cause a lot of end-point contention.

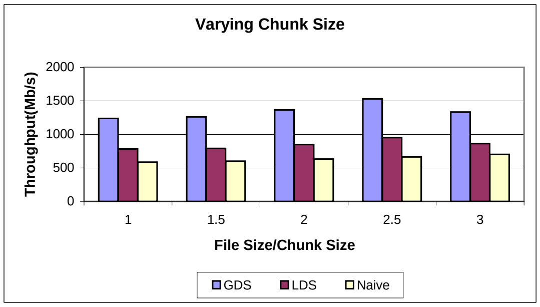


Figure 5. Performance of all the algorithms with decreasing chunk size.

Figure 4(b) shows the the relative performance of the various scheduling schemes on workloads with nodes belonging to different sites acting as destinations. This experiment was conducted across the 4 sites (BMI-ORNL-CSE-ANL) in a (4-3-3-2) configuration. The input request set consisted of around 450 file transfers, the destination nodes for each file transfer were evenly distributed across the BMI, ORNL and CSE nodes. In this case, initially, the replicas were placed only on the ORNL nodes. Then, the degree of replication is increased by placing more file replicas on the ANL and CSE nodes. As is seen from the figure, as the number of replicas is increased, the performance gap between `GDS` and the other algorithms increases. An increase in the number of replicas (with replicas being added to the CSE cluster) creates more opportunity for faster transfers and more parallelism. `LDS` and `Naive Scheduling`, however, experience a lot of end-point contention, since each node can possible act as a source and a destination for multiple files simultaneously.

In the experimental results shown so far, the replica selection algorithm only chose fully-written files as replica sources for getting portions of files. In other words, a file which is in the process of being written to a destination node v_d cannot act as a source for the transfer of the same file to another node $v_{d'}$, even if the required portion of the file at the destination $v_{d'}$ has already been written at node v_d. Once the file is completely written at node v_d, it can act as a replica source for other file transfers. We relaxed this constraint to allow for chunk-level replica sources. That is, a file which has not been completely written to a node v_d can still act as a source for other transfers of the same file. Figure 5 shows the performance results as a function of decreasing chunk size for all three algorithms by incorporating chunk-level replica sources. Here, the x-axis denotes the fraction $\frac{FileSize}{ChunkSize}$. Increasing the value of this parameter implies the file is transferred in smaller and smaller chunks. The results show that the throughput increases by employing smaller chunks up to a certain point, after which it shows a decrease. The initial increase is due to the fact that as the chunk size decreases, the number of possible replica sources for each file increases. Since each file has multiple destinations, chunks of file being written to some destination nodes can act as sources for other destination nodes. However, as the chunk size decreases further, the latency and I/O overheads of transferring the file in a greater number of chunks offset the potential benefit due to an increased number of file replicas.

5.3 Scheduling overhead

In our system, the scheduler computes the schedule information for a chunk request and sends this information to the corresponding destination node. The destination node starts the retrieval of the chunk from the source nodes. The scheduler moves on to the next pending file transfer request and repeats the whole process. Therefore, the scheduling performed by the centralized master and the file transfers between slave nodes occur in parallel. The end-to-end execution time is defined as the elapsed time between the instant when the scheduler accepts a batch of requests to the instant when all the requests have been completed. The non-overlapped scheduling time is the difference between the end-to-end execution time and the total file transfer time. In other words, the non-overlapped scheduling time is the perceived scheduling overhead. In our experiments, we observed that the non-overlapped scheduling time is negligible. This is because, the schedule is generated iteratively while the file transfers are taking place.

5.4 Lower-bound Comparisons

Tables 2 and 3 show the comparison of the lower bounds obtained from the IP formulation in Section 3 with the experimental values obtained by employing the

N	CSE-ORNL-ANL (Single dest.)			ORNL-ANL-CSE (Single dest.)		
	Lower bound	GDS	% Increase	Lower bound	GDS	% Increase
1	148.6	201.4	36	250	289.3	16
2	142.6	193.5	36	163.2	195.9	20
3	134.6	191.6	42	125.4	165.65	32
4	134.6	183.4	36	114.2	149.7	31
5	134.6	157.8	17	79.4	125.12	58

Table 2. Comparison (in terms of transfer time(secs)) between lower bounds and GDS scheduling algorithm for single-destination workloads. Here N represents the average number of initial file replicas.

N	CSE-ORNL-ANL (Multiple dest.)			ORNL-ANL-CSE (Multiple dest.)		
	Lower bound	GDS	% Increase	Lower bound	GDS	% Increase
1	347.6	611.6	76	508.8	783.44	54
2	323.5	591.5	83	295.3	580.4	96
3	322.6	585.08	81	196.3	387.7	97
4	307.6	515.8	68	116.5	252.3	117
5	307.6	508.01	65	81.5	165.33	103

Table 3. Comparison (in terms of transfer time(secs)) between lower bounds and GDS scheduling algorithm for multi-destination workloads. Here N represents the average number of initial file replicas.

GDS scheduling algorithm for single-destination workloads and multiple-destination workloads. A single-destination workload, here, refers to a workload where each file has a single destination. A multi-destination workload, on the other hand, is one in which each file is transferred to multiple destination nodes. The multi-destination workloads employed here are the same as the ones which have been used for the results shown in Figures 2 and 3. The lower bounds have been computed by employing peak values of bandwidth on the various network links. The results show that the GDS scheduling algorithm results in between 16-58% increase in execution time compared to the lower bound for the single-destination workloads and between 54-117% increase for the multiple-destination workloads. The difference between the lower bound and the GDS is attributed to the fact that observable network bandwidth over the wide-area can show fluctuations over time. Also, because of the slow-start mechanism of TCP, some file transfers cannot observe the achievable network bandwidth. The difference between the lower bound and GDS is higher in the multi-destination case as compared to the single-destination case. The IP formulation can yield solutions which employ multiple destinations v_1, v_2, ...v_{k-1} of a file to send flow to another destination v_k. However, the formulation does not capture if the sources v_1, v_2, ...v_{k-1} have the required chunks of files or not at a given instant. In the worst case, all the sources might have the same chunks of file at a given instant, which means that only a single source would be used to transfer the chunk. The IP formulation is oblivious to this since it is based on static flow concepts and does not incorporate the notion of time. Therefore, it results in an overestimation of the achievable throughput and a lower transfer time. Note that since the GridFTP servers do not have the capability to route the incoming data to a different GridFTP server, we do not allow this in the IP formulation as well. Using a GridFTP server as an intermediate node without storing the data on to the disk is non-trivial and require changes/additions to the GridFTP code and is a part of our future work.

5.5 Discussion

In this section, we provide insights into the scenarios where in our proposed algorithm is expected to provide significant performance improvements as well as those cases where it is expected to give little performance benefits. In general, the algorithm is expected to provide greater benefits with increasing degree of data replication. With a very low degree of replication, the proposed algorithm is restricted in its choice of multiple replicas, thereby not giving significant performance improvements. Moreover, the algorithm is expected to perform well for multi-destination workloads. This is because, the algorithm, can dynamically take into account the existence of new replicas as some of the files are transferred to their respective destination nodes, and employ those replicas for subsequent file transfers.

6 Conclusions

This paper proposes a dynamic scheduling algorithm which schedules a set of data transfer requests made by a batch of data-intensive tasks in a wide-area environment like the Grid. It also proposes a network flow based integer programming formulation of the scheduling problem, which is used to find a lower bound on the transfer time under idealistic conditions of resource availability and performance. The proposed dynamic scheduling algorithm is adaptive in that it accounts for network bandwidth fluctuations in the wide-area environment. The algorithm incorporates simultaneous transfer of disjoint chunks of the same file from different replica sources to a destination node, thereby increasing the aggregate bandwidth. Adaptive replica selection is used for transferring different chunks of the same file by taking dynamic network information into account. We employ GridFTP for data transfers and utilize information from past GridFTP transfers to perform predictive bandwidth estimations. We have shown the effectiveness of our scheme through experimental evaluations on a wide-area testbed.

References

[1] The Large Haldron Collider (LHC) . http://lhc.web.cern.ch/lhc/.

[2] Globus ftp client api. *http://www.globus.org/api/c/globus_ftp_client/html/index.html*, 2002.

[3] Globus metrics, version 1.4. *http://incubator.globus.org/metrics/reports/2007-02.pdf*, 2007.

[4] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp framework and server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2005. IEEE Computer Society.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, New York, NY, USA, 2003. ACM.

[6] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files from distributed repositories. In *Euro-Par 2004: Parallel Processing: 10th International Euro-Par Conference, volume 3149 of LNCS*, pages 246–253, Sept. 2004.

[7] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonindentical processors. *Journal of the ACM*, 24(2):280–289, Apr 1977.

[8] G. Khanna, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Scheduling file transfers for data-intensive jobs on heterogeneous clusters. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par*, volume 4641 of *Lecture Notes in Computer Science*, pages 214–223. Springer, 2007.

[9] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. A hypergraph partitioning based approach for scheduling of tasks with batch-shared i/o. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 792–799, Washington, DC, USA, 2005. IEEE Computer Society.

[10] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *ICDCS '04: Proc.of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 342–349, Washington, DC, USA, 2004. IEEE Computer Society.

[11] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 282–297, New York, NY, USA, 2003. ACM.

[12] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In M. Castro and R. van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2005.

[13] M. Piatek, T. Isdal, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent? In *NSDI*. USENIX, 2007.

[14] M. Swany. Improving throughput for grid applications with network logistics. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 23, Washington, DC, USA, 2004. IEEE Computer Society.

[15] TeraGrid. http://www.teragrid.org.

[16] L. Yang, J. M. Schopf, and I. Foster. Improving parallel data transfer times using predicted variances in shared networks. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 734–742, Washington, DC, USA, 2005. IEEE Computer Society.