Using Overlays For Efficient Data Transfer Over Shared Wide-Area Networks [*]

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

Data-intensive applications frequently transfer large amounts of data over wide-area networks. The performance achieved in such settings can often be improved by routing data via intermediate nodes chosen to increase aggregate bandwidth. We explore the benefits of overlay network approaches by designing and implementing a service-oriented architecture that incorporates two key optimizations – multi-hop path splitting and multi-pathing – within the GridFTP file transfer protocol. We develop a file transfer scheduling algorithm that incorporates the two optimizations in conjunction with the use of available file replicas. The algorithm makes use of information from past GridFTP transfers to estimate network bandwidths and resource availability. The effectiveness of these optimizations is evaluated using several application file transfer patterns: one-to-all broadcast, all-to-one gather, and data redistribution, on a wide-area testbed. The experimental results show that our architecture and algorithm achieve significant performance improvement.

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. For example, with the Large Haldron Collider (LHC) [3] at CERN,

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

data generated by a CMS experiment [13] must be transferred to the Tier-1 site in USA, where it is processed and then multi-cast onto many domestic Tier-2 sites. As another example, consider a multi-institutional study that collects and analyzes Gigabyte-scale biomedical image data, obtained from high-resolution scanners, to develop animal models of phenotypic characteristics of disease progression. Hundreds or thousands of images can be obtained from an animal and there can be hundreds of animals in a study. Images in multi-institutional studies may be collected and stored at multiple sites. Researchers wishing to carry out an analysis using images from many 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 data from files at the storage sites to the computational sites.

High-bandwidth, high-latency optical networks are being increasingly used by researchers and scientists. These networks enable the transfer of extremely large files with sizes up to a few petabytes. A file transfer mechanism which can optimize the overlay routes used to transfer files and take advantage of the available network parallelism can enhance the data-transfer throughput achieved by an application. In addition, a lot of scientific experiments may involve the transfer of data over public, shared networks, instead of a dedicated network infrastructure. Here it is important for the file transfer mechanism to make intelligent use of available paths to maximize the achievable bandwidth.

GridFTP [6] 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. By default, GridFTP employs TCP as the underlying transport protocol. Multiple TCP streams can be created between the source and the destination in order to offset the network congestion and improve throughput. The use of multiple streams in parallel, however, does not affect the routing or take into account network parallelism.

In this work, we seek to explore the effects of multi-hop path splitting and multi-pathing to improve the file transfer performance in GridFTP. Multi-hop path splitting improves performance by replacing a direct TCP connection between the source and destination by a multi-hop chain through some intermediate nodes. Multi-pathing involves striping the data at the source and sending it across multiple overlay paths thereby leading to a better achievable throughput. In other words, multiple independent routes can be employed to simultaneously transfer disjoint chunks of a file to its destination. We propose a path determination heuristic which incorporates these optimizations for efficient transfer of a single file. To optimize performance for batch file transfer requests, we extend a collective file-transfer scheduling heuristic implemented in an earlier work [16]. The extended algorithm incorporates multi-hop path splitting and multi-pathing optimizations.

We experimentally evaluate the optimizations and their GridFTP implementation on a wide-area testbed. Our performance metric in this work is the total transfer time of a batch of file transfer requests. We investigate performance improvements under several different file transfer patterns: one-to-all broadcast, all-to-one gather and data redistribution patterns common in many application scenarios. As an example, a one-to-all communication pattern may occur in high-energy physics, where the data stored at a US Tier-1 site needs to be broadcast to all Tier-2 sites. As another example, a all-to-one gather operation may arise when a researcher runs a biomedical image analysis on a local machine using a subset of images collected at different sites. Our results show that the proposed optimizations lead to significant performance improvements for communication patterns like one-to-all, gather, and data redistribution from a set of source nodes to a set of destination nodes. However, we also observe that the improvements are significantly lower when data replication is employed.

The rest of the paper is organized as follows. Section 2 presents an overview of the related work in wide-area data-staging. Section 3 talks about the two key optimizations we propose to use in this work, namely, multi-hop path splitting and multi-pathing. Furthermore, it also talks about our proposed path determination heuristics and a collective file-transfer scheduling framework for efficient transfer of files over the wide-area. Section 4 focuses on the design and implementation of a service-oriented architecture that incorporates the two optimizations within GridFTP file transfer protocol. Section 5 describes an experimental evaluation of our proposed heuristics.

2 Related Work

The Distributed Parallel Storage Server (DPSS) [25, 26] is a wide-area distributed storage system that employs optimizations such as parallel TCP streams and TCP tuning. It employs a collection of distributed disk servers to provide high speed random access to large-scale data. GridFTP [6] is a widely used protocol which enables secure, reliable and high performance data movement. GridFTP applies concepts from DPSS to facilitate efficient data transfer between end-systems. It employs techniques like multiple TCP streams per transfer, striped transfers from a set of hosts to another set of hosts and partial file transfers. However, GridFTP currently does not incorporate optimizations which affect network routing or take into account any network parallelism. In this work, our contribution is to explore the use of two key optimizations, namely, multi-hop path splitting and multi-pathing and propose optimization algorithms which can exploit these optimizations to maximize file transfer throughput. We design and implement these optimizations within the realm of GridFTP which is a commonly used file transfer mechanism over the wide-area.

Cai et al.[11] propose PGOS, a routing and scheduling algorithm which tries to achieve desired QoS levels for multiple competing flows. 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 achieving each competing request's desired QoS. To achieve this goal, we integrate multi-hop path splitting and multi-pathing and incorporate it in our existing collective scheduling framework [16]. Stork [19] is a specialized scheduler for data placement activities on the Grid. The scheduler allows check-pointing 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. LDR [2] is a tool for creating copies of datasets across a Virtual Organization. It involves replication of data to multiple sites as well as selection of replicas for the purpose of transferring data to a client. The data transfer optimizations proposed in this work are complementary to a framework like LDR and can be incorporated into it.

Swany [23] exploits the ”logistical effect” which essentially means improving performance by dividing a connection into a series of shorter, better performing connections. In a recent work, Rizk et al. [21] have looked at providing TCP splitting functionality with respect to GridFTP and showed performance improvement for single file transfers. In this work, we propose to optimize single file transfers by employing both multi-hop path splitting and multi-pathing in an integrated fashion. In addition, we incorporate these optimizations into a collective file-transfer scheduling framework and evaluate its effectiveness. BitTorrent 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. The optimizations that we explore in this work, can be used to improve the performance of a single file transfer of a particular file. BitTorrent is not meant to improve the performance of single file transfers, but is more suited to a case where a bunch of peers request the same file(s), that is, swarm optimization. In addition, BitTorrent does not perform any explicit multi-swarm optimization. Moreover, our goal is to perform makespan optimization where the global objective of minimizing the time is more important than each site's local benefits.

Giersch et al. [12] 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 [14] to lower the scheduling cost. In our past work, we looked at the problem of scheduling a batch of data-intensive tasks [17]. We have also investigated scheduling of file transfers in data center environments where in the scheduler has ultimate control [15]. In this work, we are targeting dynamic heterogeneous wide-area environments like Grids. In our recent work [16], we proposed a scheduling algorithm which schedules a set of data transfer requests in a wide-area environment like the Grid. The scheduling scheme, however, does not incorporate multi-hop path splitting or multi-pathing. In this paper, we propose a new dynamic scheduling algorithm which builds upon the algorithm proposed in our previous work by incorporating the ideas of multi-hop path splitting and multi-pathing.

Network Weather Service (NWS) [27] is a well-known tool used to monitor network resources. NWS relies on active “probing” to determine the available bandwidth and latencies of the actual network. However, past research has shown that NWS is not quite effective for ascertaining the performance of Grid Data transfers [5]. In this paper, we employ passive monitoring through actual data measurements, as opposed to active monitoring using a tool like NWS.

3 Data Transport Optimizations

In this section, we provide an overview of the optimizations investigated in this work.

3.1 Multi-Hop Path Splitting

The observed throughput of GridFTP transfers in a wide-area environment may be lower than the maximum achievable throughput, due to a number of factors, such as the slow-start and congestion control mechanisms used by TCP. The technique of dividing a TCP connection into a set of shorter, better performing connections by splitting it at multiple intermediate points with the goal of improving the overall throughput has been studied [8, 9, 10, 21]. A split-TCP connection may perform better than a single end-to-end TCP connection due to several reasons. First, the round-trip time on each intermediate hop is shorter as compared to the direct end-to-end path. The congestion control mechanism of TCP would sense the maximum throughput quickly thereby attaining steady state, wherein it will give maximal possible throughput until a congestion event occurs. Second, any packet loss is not propagated all the way back to the source but only to the previous intermediate hop. In this work, as an alternative to a direct TCP connection between a source and destination, we explore the use of multi-hop pipelined transfers using intermediate nodes. If the bandwidth on each of the intermediate hops is higher than the direct path, the overall throughput can be expected to improve. Figure 1 illustrates the use of multi-hop paths to transfer a file from a source to a destination. The direct path from the node C1 to C4 has the bandwidth B_{14} while an alternate path from the node C1 to C4 comprises of three hops, C1-C2, C2-C3 and C3-C4. The bandwidth on the multi-hop path B_{split} equals the minimum of the bandwidth on each of the hops.

$$B_{split} = \min\left(B_{12}, B_{23}, B_{34}\right) \tag{1}$$

The multi-hop path is preferable for transfer of data from node C1 to C4 if the B_{split} is greater than B_{14} and the end-to-end latencies of the multi-hop path and the direct path are comparable. We also refer to the alternate multi-hop path as a “split” path.

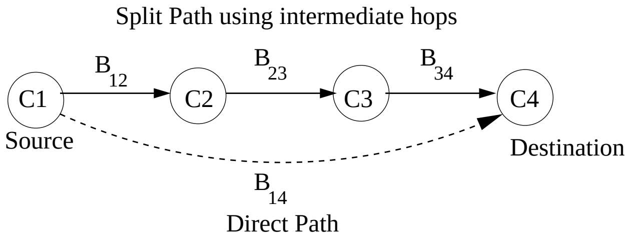
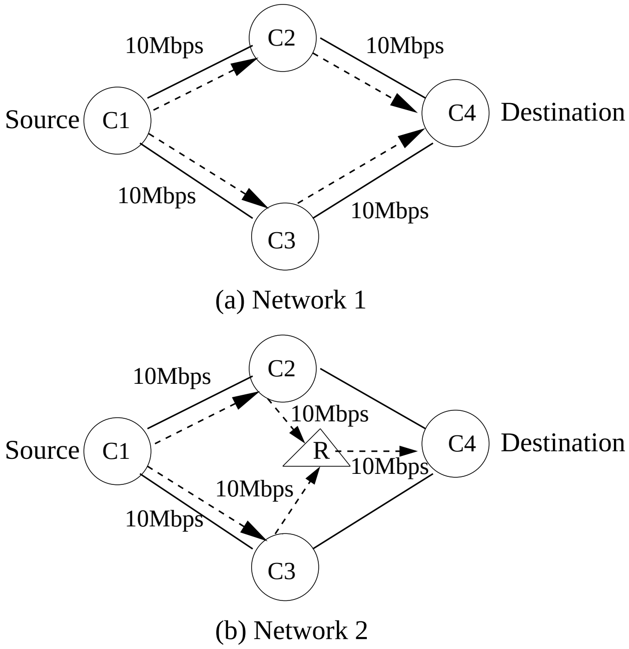


Figure 1: A multi-hop path C1-C2-C3-C4 can be used to transfer a file from C1 to C4 in a pipelined fashion.

3.2 Multi-Pathing

Striping the data at the source and sending it across multiple overlay paths can also lead to a better achievable throughput. In other words, multiple independent routes can be employed to simultaneously transfer disjoint chunks of a file to its destination.



(a) Network 1

(b) Network 2

Figure 2: (a) Network 1 with two independent paths between C1 and C4. (b) Network 2 where the paths between C1 and C4 share a common bottleneck.

Splitting a file at the source and transferring it through multiple independent routes is equivalent to solving the maximum flow problem in a graph, where the graph vertices represent the overlay nodes (e.g., GridFTP servers), and the edge capacities equate to the network bandwidth between the corresponding pair of nodes. However, a direct application of maximum flow concepts does not account for bottlenecks due to physical sharing of links and routers. For example, in Figure 2(a), two independent paths exist between C1 and C4. Therefore, the overall bandwidth for file transfers between C1 and C4 can be increased by utilizing the two paths simultaneously provided the hosts C1 and C4 can handle the combined data rate. However, if there is a shared link which becomes bottleneck, the overall bandwidth would not increase by increasing the number of overlay transfers. In Figure 2(b) an additional routing node, R, is present, which is not a part of the overlay network. The routes C1-C2-C4 and C1-C3-C4 both share the bottleneck link R-C4. Therefore, an approach that finds multiple parallel links, but does not consider the physical "underlay" network, will find suboptimal solutions. This is key to maximizing the effective aggregate bandwidth.

3.3 Path Determination Algorithm

The path determination algorithm (Algorithm 1) is an iterative algorithm that computes a set of paths which can be collectively used to transfer a file from its source node(s) to its destination node. At each step, algorithm invokes Best Path heuristic (Algorithm 2) to find a path that will yield minimum transfer time for the requested file given the concurrent transfers in the overlay network. It then modifies the overlay network graph to reflect the current transfer, and continues it search for another path. Since past research has shown that most of the benefit that can be obtained by splitting TCP can be achieved by using up to 2 hops and adding extra hops does not yield significant benefit [20], in this algorithm, we only consider paths of length 2 or 3 when looking at multi-hop paths.

In this work, the wide-area environment is represented by a graph $G = (V, E)$, referred as the *platform graph*. In the platform 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 w_{ij} is a measure of the achievable bandwidth BW_{ij} between the two machines. Let RTT_{ij} be the round-trip time of the wide-area path between the nodes v_i and v_j. The best path heuristic takes as input the overlay Graph $G = (V, E)$, the set of the sources V_s of the file f_ℓ, and the destination node v_d.

The best path to transfer a file from the source or a set of sources to a destination, is the path which yields the minimum expected completion time to transfer the file. The best path heuristic is a variant of the Dijsktra's shortest path algorithm. The algorithm involves creating a new Graph $G' = (V', E')$ where $V' = V$ and $E' = E$. However, the weighting function employed for weight assignment to an edge between the nodes v_i and the node v_j is the ratio of the round-trip time of the path corresponding to the edge to its bandwidth. The motivation behind this is to give preference to low-latency high-bandwidth edges. The other difference is the calculation of the distance function to measure the goodness of a path. Since the transfer of a file from a source node to a destination node through a multi-hop path occurs in a pipelined fashion, therefore, the distance function of a path is computed as the maximum of the weights on each of its constituent edges (see step 18). Note that the traditional shortest path algorithm employs the distance function to be the sum of weights instead of the maximum.

Algorithm 1 Path Determination

```
Input: Platform $G = (V, E)$, request $< f_\ell, v_d >$, where the file $f_\ell$ is requested by destination $v_d$ and a set $V_s$
    representing the set of sources of the file $f_\ell$.
$Chosen\ Paths = \emptyset$
while 1 do
    Run the Best Path Heuristic. Let $MinPath$ be the output.
    Let $MinWeight$ be the minimum weight edge on the path $MinPath$.
    if $MinWeight == 0$ then
        return
    else
        Add the path $MinPath$ to the set $Chosen\ Paths$.
        for Each edge $(v_i, v_j) \in MinPath$ do
            $w_{ij} = w_{ij} - MinWeight$
return $Chosen\ Paths$
```

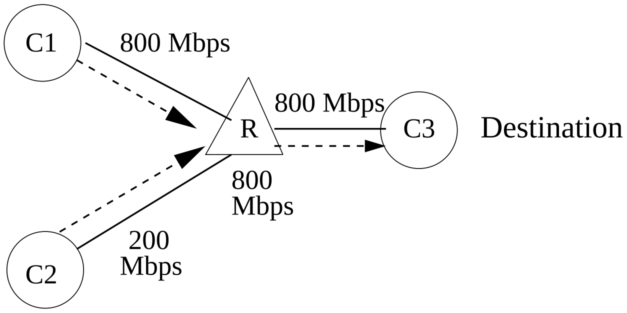


Figure 3: An example setup with shared resources.

3.4 Modeling Bottleneck due to Shared Resources

The path determination algorithm presented in Section 3.3 chooses a set of independent paths to collectively transfer a file. However, one or more selected paths can possibly share bottleneck links, which means that the overall bandwidth would not necessarily increase by employing multiple paths. In some cases, the aggregate bandwidth might sometimes even decrease by employing multiple paths. An example of a setting with shared routers and links is illustrated in Figure 3. In this setting, two paths, C1-R-C3 and C2-R-C3, can be simultaneously utilized to transfer data. If the existence of router R is oblivious to the multi-pathing decision algorithm, then it will choose to split a file of size say 1000 Mb into two parts, one of size 800 Mb which is transferred along the path C1-R-C3, and the other of size 200 Mb which is transferred along the path C2-R-C3. Since the router R can only sustain a bandwidth of 800Mbps, the flow along the path C1-R-C3 will saturate R. In that case, the two flows are effectively serialized, requiring 2 seconds to transfer the file. The aggregate bandwidth, therefore, is 500Mbps. On the other hand, a multi-pathing decision, which incorporates the knowledge of the existence of R, can choose to send the entire flow along the path C1-R-C3, thereby getting a throughput of 800Mbps.

We model the shared bottlenecks by performing an offline characterization of the network. For each pair of edges

Algorithm 2 Best Path

```
Input: Platform $G = (V, E)$, request $< f_\ell, v_d >$, where the file $f_\ell$ is requested by destination $v_d$ and a set $V_s$
    representing the set of sources of the file $f_\ell$.
$Output\ Path = \emptyset$
Create a new graph $G' = (V', E')$ with $V' = V$ and $E' = E$. Weight of an edge $w_{ij}$ in the Graph $G'$ equals $\frac{RTT_{ij}}{NetBW_{ij}}$.
Define a variable $Dist_i$ for each vertex.
Define a variable $HopCount_i$ for each vertex.
Define a variable $Pred_i$ for each vertex.
for each node $v_i \in V'$ do
  if $(v_i \in V_s)$ then
    $Dist_i$ = 0
  else
    $Dist_i = \infty$
for each node $v_i \in V'$ do
  $HopCount_i$ = 0
  $Pred_i$ = -1
Unmark all vertices of the Graph $G'$
while There exists an unmarked vertex in $G'$ do
  Pick the unmarked vertex $v_i$ with the minimum value of $Dist_i$ among all unmarked vertices.
  for Each vertex $v_j$ adjacent to $v_i$ do
    if $(Dist_j \geq \max(Dist_i, w_{ij})) \wedge ((HopCount_i \leq 2 \wedge v_j == v_d) \vee (HopCount_i < 2 \wedge v_j \neq v_d))$ then
      $Dist_j = \max(Dist_i, w_{ij})$
      $Pred_j = i$
      $HopCount_j = HopCount_i + 1$
  Set the vertex $v_i$ as marked.
$Output\ Path = v_d$
$split = Pred_d$
while $split \neq -1$ do
  Prepend $split$ to $Output\ Path$
  $split = Pred_{split}$
return $Output\ Path$
```

$< e_1, e_2 >$ in the graph G, we first measure the end-to-end throughput for the wide-area paths corresponding to edges e_1 and e_2 with no interference. Then we measure the end-to-end throughput on the two edges by performing file transfers along them in parallel. The measured throughput values along e_1 and e_2 are then used to figure out if the two edges share a common bottleneck. The output generated by this analysis is a set of two-tuples $Shared$, where each element of the set $Shared$ represents a set of edges in the overlay which share a common bottleneck. For example, if $Shared = \{< e_1, e_2 >, < e_3, e_4 >\}$, it means that edges e_1 and e_2 share a common bottleneck and so do the edges e_3 and e_4. We ran the traceroute utility on the elements of the set $Shared$ and verified that the edges indeed involved shared links.

The offline characterization corresponding to identification of shared bottlenecks is then used to avoid choosing multiple overlay paths wherein the aggregate bandwidth would not increase. In future, we plan to employ more sophisticated techniques [18], [22] for more accurate characterization of underlying congestion.

Algorithm 3 shows a variant of the proposed Path Determination algorithm which incorporates this knowledge.

Algorithm 3 Path Determination with modeling of shared bottleneck

```
Input: Platform $G = (V, E)$, request $< f_\ell, v_d >$, where the file $f_\ell$ is requested by destination $v_d$ and a set $V_s$
    representing the set of sources of the file $f_\ell$.
$Chosen\ Paths = \emptyset$
while 1 do
  Run the Best Path Heuristic. Let the path returned be represented by $MinPath$.
  Let MinWeight be the minimum weight edge on the path $MinPath$.
  if MinWeight == 0 then
    return
  else
    for Each Path $APath \in Chosen\ Paths$ do
      if $APath$ and $MinPath$ share an underlying bottleneck then
        return
    Add the path $MinPath$ to the set of selected paths $Chosen\ Paths$.
    for Each edge $(v_i, v_j) \in MinPath$ do
      $w_{ij} = w_{ij} - MinWeight$
return $Chosen\ Paths$
```

3.5 Scheduling a Batch of File Transfers

The optimizations presented in Sections 3.1 – 3.4 aim to improve performance for single-file transfers. In some applications, a batch of file transfers need to be handled. In a recent work [16], we proposed a dynamic scheduling algorithm which schedules a set of file transfer requests made by a batch of data-intensive tasks in a wide-area environment. The scheduling algorithm is iterative, employs adaptive replica selection, and makes 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. The scheduling scheme, however, did not incorporate multi-hop path splitting or multi-pathing. The path to transfer a file from a source node to a destination node follows the underlying network routing. In this paper, we propose a new dynamic scheduling algorithm which builds upon the algorithm proposed in our previous work by incorporating the ideas of multi-hop path splitting and multi-pathing.

This scheduling scheme proceeds in steps and in each step it selects a pending file transfer request $< f_\ell, v_d >$ from the request list, R, and computes a schedule for the request. The schedule for a request consists of the set of paths to be employed to collectively transfer the file.

In our current implementation, we employ Globus GridFTP as the underlying transfer mechanism [7]. Each source node runs a Globus 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 file has been computed, the scheduler sends the schedule information to the corresponding destination node. The destination node starts the retrieval of the file 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 4.

Algorithm 4 Global Dynamic Scheduling with Multi-Hop Path Splitting and Multi-Pathing

```
Input: Platform $G = (V, E)$ and a set $R = \{< f_\ell, v_d > \mid$ file $f_\ell$ is requested by destination $v_d\}$
 1: $HostBw_i =$ the host bandwidth at node $v_i$
 2: while there are pending requests, i.e., $R \neq \emptyset$ do
 3:   if $\exists v_d$ such that $HostBw_d > \epsilon$ then
 4:     for each request $r =< f_\ell, v_d >\in R$ do
 5:       Create a new graph $G'$ identical to $G$.
 6:       $ChosenPaths \leftarrow$ Path Determination$(G', r)$
 7:       Compute the expected finish time to transfer the file $f_\ell$ to destination $v_d$.
 8:     Choose the request $r$ with the minimum expected finish time
 9:     Schedule the transfer of the file along the selected paths to the node $v_d$.
10:     $R \leftarrow R - \{r\}$
11:     Update the expected available host bandwidth ($HostBw_i$) along the paths chosen to complete the request.
12:   for every finished file transfer request $< f_\ell, v_d >$ do
13:     Update the available network bandwidths along the paths chosen to complete the request
```

At step 6, the Path Determination method is invoked to select multiple paths for the transfer request. The output from this method makes up the schedule for the request. The next step (step 7) 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 file into K portions where K is the number of selected paths. The size of the portion is chosen to be in the same ratio as that of the bottleneck bandwidth of the paths. The transfer completion time is then simply the maximum of the times taken to send each portion along its designated path. At step 9, following the well-known Min-Min [14] 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 scheduler then updates the value of available bandwidths at the nodes along the chosen paths for the scheduled request. The overall process repeats until all the file transfers have been scheduled.

4 GridFTP with Path Splitting and Multi-Pathing

We now discuss the design and implementation of a modified GridFTP server/client infrastructure, which incorporates the multi-hop path splitting and multi-pathing optimizations for file transfer. Our goal is to design a framework which will allow GridFTP clients to benefit from path splitting and multi-pathing in a transparent manner, i.e., clients will issue GridFTP file transfer commands in much the same way as they do now.

The path splitting and multi pathing optimizations require information about the overlay graph of GridFTP servers in order to make efficient routing decisions and to leverage existing “network parallelism” by using multiple data flows on distinct paths. Obtaining the overlay graph information for very large grids is a hard problem especially since the grids do not have a centralized authority. However, the usage of multiple paths to transfer a file from a source to a destination through an overlay network of GridFTP servers is more suited to a scenario when those GridFTP servers are involved in a collaborative setting (e.g. a scientific experiment like LHC at CERN) and are willing to provide their resources for forwarding of data to other sites. In other words, the applicability of these concepts is more suitable in a virtual organization (VO) like setting.

We describe a decentralized system to construct and maintain the overlay graph of GridFTP servers in Section 4.2. Our optimized GridFTP client first contacts the source GridFTP server (specified by the user) and retrieves the overlay graph information. The overlay information is basically a set of tuples $< source, destination, bandwidth >$. Once the GridFTP client has the required information, it runs the path determination algorithm algorithm to determine the multi-hop path(s) through which the file is to be transferred. Implementation of path-splitting in GridFTP involves the modification of the GridFTP server since it involves forwarding of data through intermediate hops. However, multi-pathing can be incorporated by just modifying the GridFTP client globus-url-copy. The GridFTP client, globus-url-copy, currently has support for accepting a set of source-destination url pairs and realizing the file transfers corresponding to them. We employ this functionality to realize multi-pathing. Once a set of paths have been determined by a decision algorithm, globus-url-copy creates a new list of urls along with partial offsets and lengths associated with each url and then performs the file transfer using the multiple designated paths to the destination. In the next section, we present how we have implemented path splitting in GridFTP.

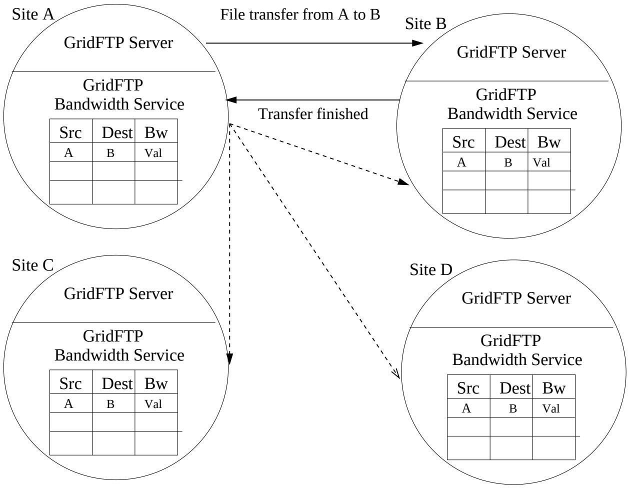


Figure 4: A set of GridFTP servers forming an overlay and sharing point-point bandwidth information.

4.1 Implementing Path Splitting in GridFTP

GridFTP uses a Data Storage Interface (DSI) to interact with the storage system. The DSI layer accepts requests (e.g., get, put and stat) and performs the required operations on the storage system. To achieve the split-TCP effect, the DSI module must be modified to perform different operations based on the data routing requirements. For instance, a GridFTP server DSI *put* could either transfer the data to the underlying disk subsystem or it could act as a split-point and transfer the data to another GridFTP server. Rizk et al. [21] have implemented a DSI interface to achieve the split-TCP functionality. Their implementation enables a GridFTP client to specify a multi-hop transfer between a source and a destination URL through a series of intermediate hosts by specifying split URLs. A split URL is essentially a concatenation of multiple normal URLs. For example, a GridFTP client command issued with a source URL A/B and the destination URL C/D means that the file will be transferred from A to D via B and C. Using this DSI for split-TCP functionality requires the user to define the overlay route required.

We have made a number of improvements to their work. In their implementation, a GridFTP server could either act as an end point or as an intermediate server but not both, which is very restrictive from the point of view of production-use GridFTP servers. We have extended the DSI layer by allowing it to simultaneously act as an end-point and an intermediate "hop" for different connections. In addition, we have also provided support for directory operations using intermediate servers, a feature not provided in their work [21]. Furthermore, their implementation worked only in the non-secure mode. We have incorporated the necessary changes to make the split-TCP work with GSI security mechanism.

4.2 Constructing the Overlay Graph

To obtain the overlay graph information, each GridFTP server acts as a resource provider, thereby exposing WS resource properties (e.g., the bandwidth achieved by file transfers using that GridFTP server). We propose a decentralized service-oriented architecture to facilitate sharing of bandwidth data among sites.

Each GridFTP site also runs a GridFTP bandwidth reporting service. The bandwidth reporting service works in conjunction with the GridFTP server running at the site. For every file transfer which happens through the GridFTP server, the GridFTP server contacts the bandwidth reporting service with the three tuple $< source, destination, bandwidth >$ associated with the transfer. The bandwidth reporting service sends the tuple to each of the peer sites which comprise the overlay. Each of the sites maintains a circular queue of bandwidth values for each source-destination pair. The circular queue stores historical bandwidth information which is used to make predictions about bandwidths in the subsequent time intervals. 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 [28] for more accurate bandwidth predictions. In this way, each of the sites maintains a set of three tuples $< source, destination, bandwidth >$ corresponding to peer GridFTP sites in the Grid. The bandwidth reporting service also exposes API which allow external entities (e.g. GridFTP clients) to access the bandwidth information and use it to optimize routing and multi-pathing decisions. We employ a Globus toolkit component C-WS core [1], to develop and deploy the bandwidth reporting service at each site.

5 Experimental Results

We compare our dynamic scheduling approach against our previously proposed work [16] as well as a baseline strategy that we refer to as *Naive Scheduling*. In this approach, each destination site picks a randomly chosen replica source for retrieving a file instead of employing dynamic bandwidth information or multiple replicas. Here onwards, we refer to our previously proposed scheduling algorithm [16] as *GDS* (Global Dynamic Scheduler), the scheduling variant that incorporates path-splitting and multi-pathing optimizations as *GDS-MHMP*, and the scheduling variant that incorporates the modeling of shared bottleneck on top of these two as *GDS-MHMP-SB*.

5.1 Experimental Setup

We use the Globus implementation of GridFTP to transfer files [7]. GridFTP exposes a set of API calls [4] for setting TCP buffer sizes and to obtain portions of a file from a source. For batch transfers, a master scheduler sends control information to clients (destination hosts). The destination hosts then call *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.

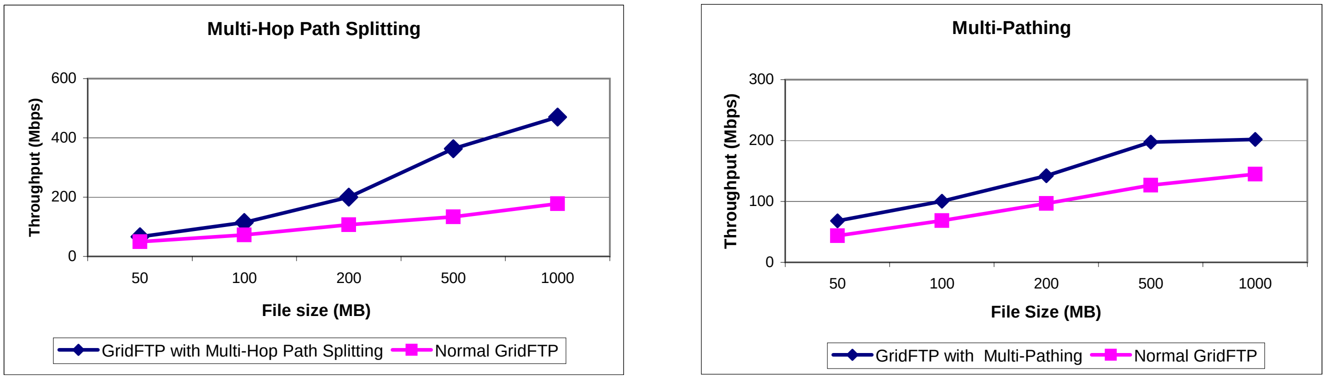
The experiments were carried out across five geographically distributed sites: BMI, a memory/storage cluster in the Department of Biomedical Informatics at the Ohio State University; ST, the Starlight site in Chicago; JA site in Japan which is a part of the Japan Gigibit Network II (JGN2) project; ORNL, which consists of 28 dual processor 3.06 GHz Intel Xeon sites; and ANL, a IA-32 Linux cluster which consists of 96 dual-processor Intel Xeon sites. The latter two sites are on the Teragrid [24] network. Table 1 shows the bandwidths in Mbps (Megabits per second) between pairs of sites of different sites.

For evaluation, we compared the performance of the various scheduling schemes under a set of well-known communication patterns. For the experimental workloads, we employed files of size 100MB.

Figures 5(a) and 5(b) highlight the performance improvement due to employing multi-hop path splitting and multi-

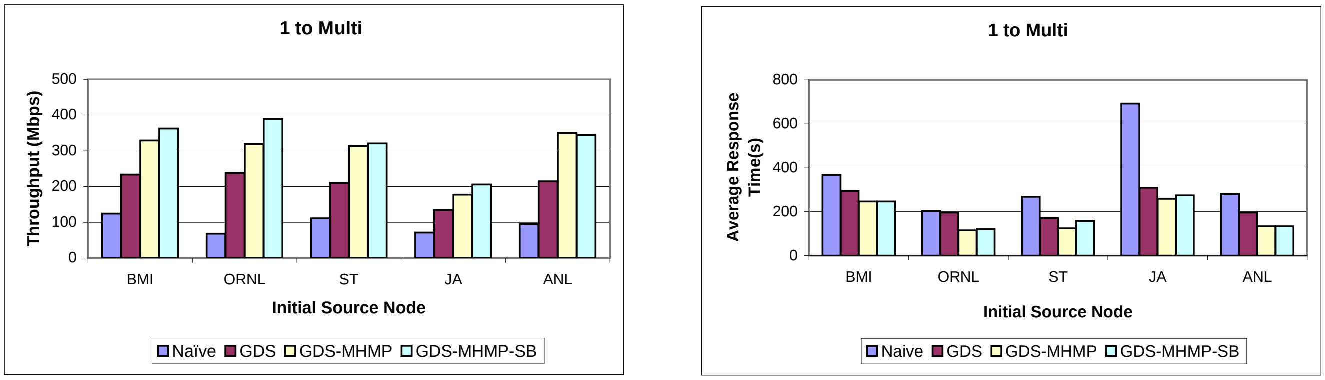
	BMI	ORNL	ST	JA	ANL
BMI	880	200	200	70	4
ORNL	200	900	800	100	20
ST	80	900	900	800	150
JA	40	120	800	900	8
ANL	60	600	500	30	900

Table 1: Link bandwidths (in Mbps) between every pair of sites.



(a) (b)

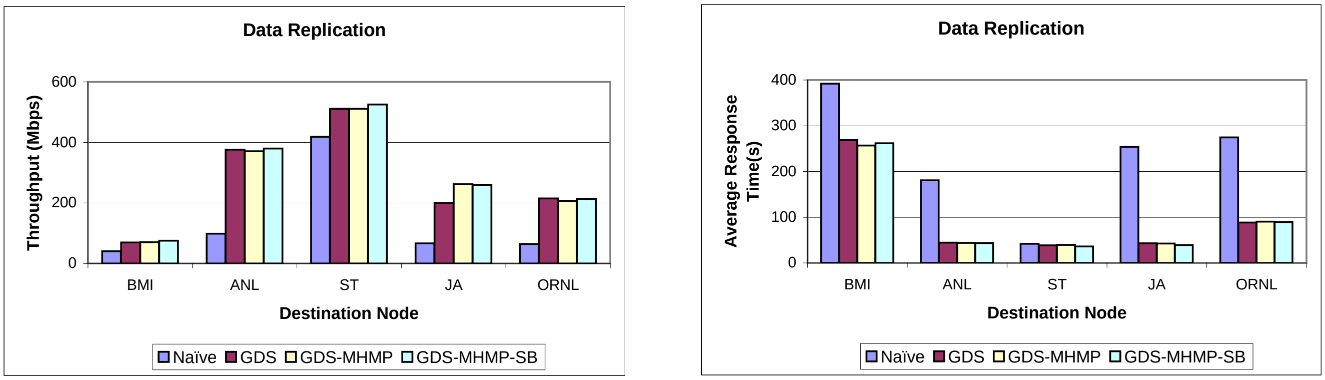
Figure 5: (a) Performance improvement due to multi-hop path splitting using the path JA-ST-ANL as compared to using the default path JA-ANL, (b) Performance improvement due to multi-pathing by employing the paths BMI-ORNL-JA and BMI-ST-JA in parallel as compared to using the default path BMI-JA.



(a) (b)

Figure 6: Performance of all the algorithms under the 1-to-all communication pattern in terms of the (a) Average throughput and (b) Average response time.

pathing, respectively. Figure 5(a) compares the performance of a file transfer from the JA site to the ANL site using the direct path as governed by underlying routing, with the case when the ST site is employed as an intermediate path-splitting site. The results show that the multi-hop path performs significantly better than the direct transfer, especially at very large file sizes. This is because, the bandwidth of both JA-ST and ST-ANL is higher as compared to JA-ANL while the end-to-end latency in both the cases is similar. Therefore, the overall throughput improves. Figure 5(b) compares the performance of a file transfer from the BMI site to the JA site using the direct path, with the case when the file is split at the BMI site and sent across two independent paths BMI-ORNL-JA and BMI-ST-JA. The results show that with multi-pathing, performance improves by up to 55%. This is because, by employing independent paths, the aggregate bandwidth at the destination site increases.



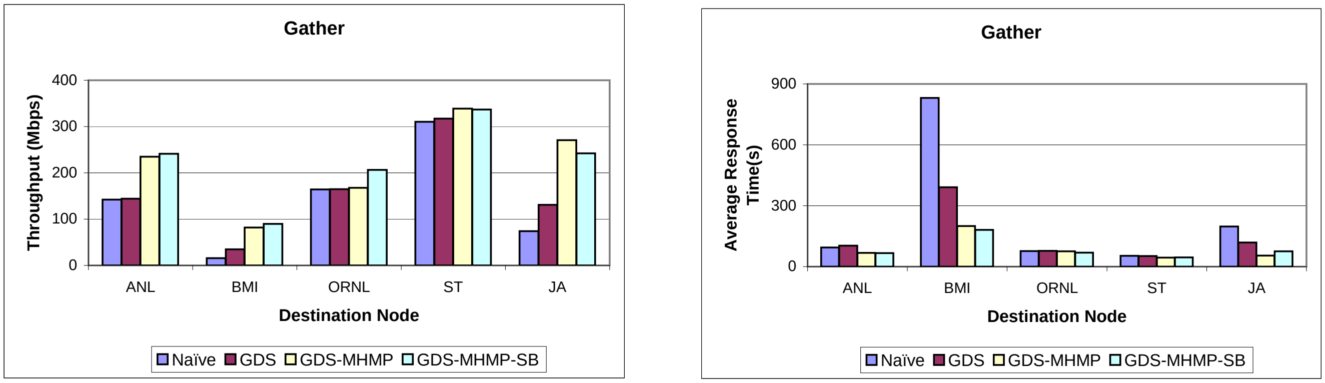
(a) (b)

Figure 7: Performance of all the algorithms under the effect of data replication, in terms of the (a) Average throughput and (b) Average response time.

Figure 6 shows the performance of the scheduling schemes under a 1-to-all communication pattern. In this experiment, only one of the 5 sites acts as a source initially and stores all the files. Each of the file needs to be transferred to all the destination sites. This experiment involved transferring 40 100MB files from the source site to each of the destination sites. The results show that *GDS-MHMP* leads to significant improvements in the achieved throughput over the *GDS*. This is because, with only a single source present initially, *GDS* is initially restricted to only using the default path between the source and the destinations. As more replicas of a file get created, those can also act as replica sources thereby giving more flexibility to *GDS*. *GDS-MHMP*, on the other hand, exploits path splitting and multi-pathing and thereby provides, significant performance improvement. An example instance of multi-hop path splitting being, the file transfer from the BMI site to the JA site via the ST site provides 60% improvement over a direct transfer from the BMI site to the JA site for the file size under consideration. Similarly, an example instance of multi-pathing is the scenario which involves the transfer of a file from the BMI site to the JA site. Multi-pathing results in transferring disjoint pieces of file simultaneously across the two paths BMI-ORNL-JA and BMI-ST-JA. The resulting performance improvement over a direct transfer over the default path is around 39%. *GDS-MHMP-SB* performs similar or better in comparison to *GDS-MHMP*. The maximum performance improvement achieved due to modeling the shared bottleneck occurs in the case when ORNL is the initial source site. In this case, the transfer of some of the files to the ST site and the ANL site (by employing ST site as the source) finish earlier and subsequent transfer of those files to the BMI site employs all the three sources in parallel, that is, the ANL site, the ST site and the ORNL site. However, Paths ORNL-BMI, ST-BMI share common bottlenecks and so do the paths ORNL-BMI and

ANL-BMI thereby leading to improved performance for *GDS-MHMP-SB*. The results also show that the proposed scheduling schemes are able to consistently outperform the *Naive Scheduling* scheduling approach. This is because, in the *Naive Scheduling* approach, clients act independently and make requests for files without any coordination. Each file needs to be sent to multiple different destinations, thereby leading to increased end-point contention due to multiple simultaneous requests for the same file. In terms of the average response time, *GDS* and its variants outperform *Naive Scheduling*. This is because, *GDS* schedules the requests with the minimum expected completion time first. On the other hand, in *Naive Scheduling*, since multiple clients act independently of each other, therefore, requests with higher expected completion times can possibly execute before requests with lower expected completion times, thereby, increasing the overall response time.

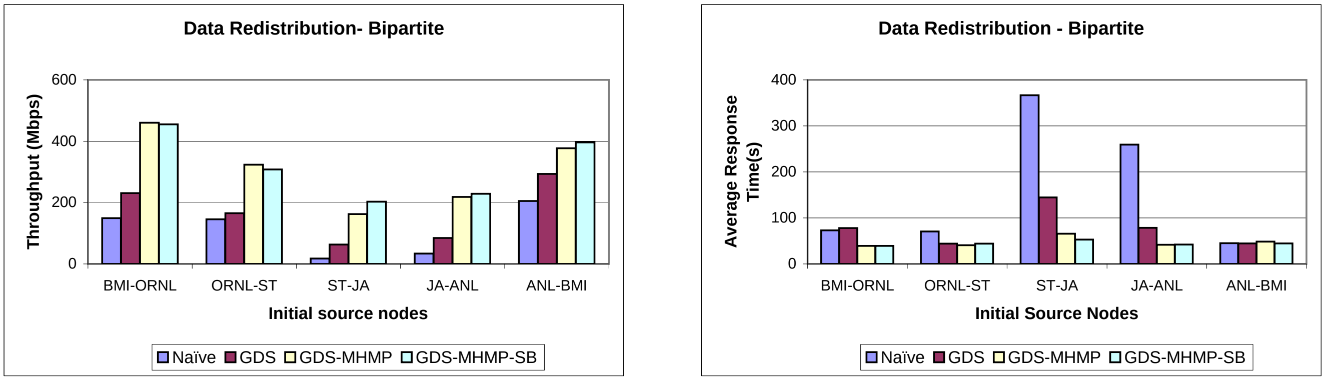
Figure 7 shows the performance of the scheduling schemes under the influence of file replication. In this experiment, each of the sites except one, stores a copy of each file. One of the sites act as the destination site to which all the files need to be transferred. Therefore, there are 4 file replicas and the fifth site acts as a destination. This experiment involved transferring 40 100MB files to the destination site. The results show that the scheduling schemes *GDS*, *GDS-MHMP* and *GDS-MHMP-SB* perform fairly similar. This happens because of the existence of multiple initial file replicas, due to which *GDS* makes the choice for the best replica to realize each file transfer. Therefore, the extent of performance improvement due to path-splitting or multi-pathing is very minimal.



(a) (b)

Figure 8: Performance of all the algorithms under a all-to-1 gather file transfer pattern in terms of the (a) Average throughput and (b) Average response time.

Figure 8 shows the performance of the scheduling schemes for a all-to-1 gather file transfer operation. In this experiment, a set of files are distributed in a round-robin fashion across 4 of the sites, and one of the sites is employed as a destination for all the files. Each file has only replica in the beginning, that is, each file is present on only one of the sites. The performance results show that in some cases, *GDS-MHMP* and *GDS-MHMP-SB* perform significantly better than *GDS* because these algorithms are able to exploit path splitting and multi-pathing to explore higher bandwidth



(a) (b)

Figure 9: Performance of all the algorithms under a bipartite data redistribution pattern in terms of the (a) Average throughput and (b) Average response time.

paths, thereby improving the throughput, while in other cases, the performance improvement is relatively less. This is because of employing a single destination site in each case, which means that the opportunities for path-splitting and multi-pathing are limited.

Figure 9 shows the performance of the scheduling schemes for a file pattern which involves data redistribution from a set of source sites to a set of destination sites. In other words, the file transfer pattern is bipartite. Files are distributed in a round-robin order on two of the 5 sites, and the data needs to be remapped to the other three sites in a round-robin manner. The results show that *GDS-MHMP* and *GDS-MHMP-SB* consistently outperform *GDS* in all the cases. This is because, multiple sites collectively act as destinations for a bunch of files, thereby creating more opportunities for path-splitting and multi-pathing in each scenario.

6 Conclusions

In this paper, we explored two key optimizations, namely, multi-hop path splitting and multi-pathing to improve the performance of file transfers over shared wide-area networks. We presented a path determination algorithm which integrates the aforesaid optimizations in order to improve achievable file transfer throughput for a single file transfer. We incorporated this with our previously proposed wide-area scheduling algorithm by making it path-splitting and multi-pathing aware. We also presented the design and implementation of service-oriented architecture which incorporates these ideas within the well known file transfer protocol, GridFTP. Finally, we looked at certain well-known communication patterns, experimentally analyzed the performance of the proposed algorithm on those patterns and show its effectiveness on a wide-area testbed. We observed that the proposed algorithm yields significant performance improvements for communication patterns like 1-to-all broadcast, all-to-1 gather, data redistribution. However, for scenarios which involve data replication, we observed that the improvements are not that significant.

References

[1] GT C WS core.
http://www.globus.org/toolkit/docs/4.0/common/cwscore/.

[2] LDR Project (2004) Lightweight Data Replicator. http://www.lsc-group.phys.uwm.edu/LDR/.

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

[4] Globus FTP Client API, 2002.
http://www.globus.org/api/c/globus_ftp_client/html/index.html.

[5] Predicting the performance of wide area data transfers. In *IPDPS '02: Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, page 34, Washington, DC, USA, 2002. IEEE Computer Society.

[6] W. Allcock. Gridftp: Protocol extensions to ftp for the grid. In *Global Grid ForumGFD-R-P.020*, 2003.

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. 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.

[8] A. Bakre and B. R. Badrinath. I-tcp: indirect tcp for mobile hosts. In *ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems*, page 136, Washington, DC, USA, 1995. IEEE Computer Society.

[9] M. Beck, T. Moore, J. S. Plank, and M. Swany. Logistical networking: Sharing more than the wires. In S. Hariri, C. A. Lee, and C. S. Raghavendra, editors, *Active Middleware Services*, Norwell, MA, 2000. Kluwer Academic.

[10] K. Brown and S. Singh. M-tcp: Tcp for mobile cellular networks. *SIGCOMM Comput. Commun. Rev.*, 27(5):19–43, 1997.

[11] Z. Cai, V. Kumar, and K. Schwan. Iq-paths: Self-regulating data streams across network overlays. In *Proceedings of the The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, 2006.

[12] 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.

[13] K. Holtman. Cms data grid system overview and requirements. In *Computing in High Energy and Nuclear Physics (CHEP)*, 2001.

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

[15] 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.

[16] G. Khanna, T. Kurc, U. Catalyurek, R. Kettimuthu, P. Sadayappan, and J. Saltz. A dynamic scheduling approach for coordinated wide-area data transfers using gridftp. In *Proc. of 22th International Parallel and Distributed Processing Symposium (IPDPS)*, Miami, Florida, 2008. to appear.

[17] 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.

[18] M. S. Kim, T. Kim, Y. Shin, S. S. Lam, and E. J. Powers. A wavelet-based approach to detect shared congestion. *SIGCOMM Comput. Commun. Rev.*, 34(4):293–306, 2004.

[19] 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.

[20] H. Pucha and Y. C. Hu. Overlay tcp: ending end-to-end transport for higher throughput. In *Poster in ACM SIGCOMM*, Philadelphia, PA, 2005.

[21] P. Rizk, C. Kiddle, and R. Simmonds. A gridftp overlay network service. In *In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, Baercelona, Spain, 2007.

[22] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 145–155, New York, NY, USA, 2000. ACM.

[23] 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.

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

[25] B. Tierney, J. Lee, L. T. Chen, H. Herzog, G. Hoo, G. Jin, and W. E. Johnston. Distributed parallel data storage systems: a scalable approach to high speed image servers. In *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pages 399–405, New York, NY, USA, 1994. ACM Press.

[26] B. L. Tierney, J. Lee, B. Crowley, M. Holding, J. Hylton, and F. L. D. Jr. A network-aware distributed storage cache for data intensive environments. In *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, page 33, Washington, DC, USA, 1999. IEEE Computer Society.

[27] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[28] 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.