Algorithms for Integrated Routing and Scheduling for Aggregating Data from Distributed Resources on a Lambda Grid

Amitabha Banerjee, *Student Member*, *IEEE*, Wu-chun Feng, *Senior Member*, *IEEE*, Dipak Ghosal, and Biswanath Mukherjee, *Fellow*, *IEEE*

Abstract—In many e-science applications, there exists an important need to aggregate information from data repositories distributed around the world. In an effort to better link these resources in a unified manner, many lambda-grid networks, which provide end-to-end dedicated optical-circuit-switched connections, have been investigated. In this context, we consider the problem of aggregating files from distributed databases at a (grid) computing node over a lambda grid. The challenge is 1) to identify routes (that is, circuits) in the lambda-grid network, along which files should be transmitted, and 2) to schedule the transfers of these files over their respective circuits. To address this challenge, we propose a hybrid approach that combines offline and online scheduling. We define the Time-Path Scheduling Problem (TPSP) for offline scheduling. We prove that TPSP is NP-complete, develop a Mixed Integer Linear Program (MILP) formulation for TPSP, and then propose a greedy approach to solve TPSP because the MILP does not scale well. We compare the performance of the greedy approach on a few representative lambda-grid network topologies. One key input to the offline schedule is the file transfer time. Due to dynamics at the receiving end host, which is hard to model precisely, the actual file transfer time may vary. We first propose a model for estimating the file transfer time. Then, we propose online reconfiguration algorithms so that as files are transferred, the offline schedule may be modified online, depending on the amount of time that it actually took to transfer the file. This helps in reducing the total time to transfer all the files, which is an important metric. To demonstrate the effectiveness of our approach, we present results on an emulated lambda-grid network testbed.

Index Terms—Lambda grid, circuit switching, routing, scheduling, large-scale data transfers.

1 Introduction

The next generation of large-scale scientific computing applications will involve expensive resources such as supercomputers, storage systems, and experimental facilities, which are distributed across domains and geographical locations. Some examples of such applications, which are being developed, include the Genomes-to-Life (GTL) Project of the US Department of Energy (DoE) [3], Teragrid [8], and the OptIPuter [6] project. Such projects typically require real-time transfer of gigabytes or petabytes of data from remote experimental sites and data warehouses across wide-area networks to a central computation site for data aggregation, processing, visualization, and other analysis. In this work, we consider applications that require centralized data computation, as opposed to a distributed approach.

These requirements are addressed by lambda-grid networks, which are backbone networks supported on the optical fiber technology. They offer an end-to-end optical circuit (also known as a wavelength or a lambda) between two end points. Such lambdas may be requested on demand or may be reserved in advance. The full bandwidth available in an optical circuit (OC-192 or 10 gibagits per second (Gbps) using current technology) may thus be made available to applications such as the GTL. A dedicated end-to-end optical channel avoids the network congestion that is typically observed in packet-switched networks (in the current Internet) and, therefore, may provide a deterministic bound on the time required for data transfer. Such a reliable and dedicated infrastructure available on demand is a key resource for the above applications. Examples of recent lambda-grid networks are the National LambdaRail (NLR) [5], DoE UltraScience Net (USN) [2] and CANARIEs CA*net [1].

We consider the problem of bandwidth reservation and scheduling on a lambda grid. As an example, the USN [2] can accept bandwidth requests for dedicated channels on demand or for future time slots (where a channel is operated in a time-division multiplexing (TDM) fashion), and it grants reservations corresponding to such requests based on feasibility constraints. We consider applications that require aggregating data from remote data sites to a centralized node before computational processing. A large number of modern e-science applications fall in this category. A specific example is the GTL application [3]. Since data is aggregated at the time of computation, the time required to transfer the data over the network may be the main computational bottleneck. Even a single second of idle time, during which the data is being aggregated, may result in the loss of several teraflops of computation power [3]. Therefore, minimizing the delay in data aggregation is the key to improve the overall system throughput.

Resource scheduling algorithms such as machine scheduling have been studied extensively in the literature: surveys on this topic can be found in [13] and [16]. Such

- *A. Banerjee, D. Ghosal, and B. Mukherjee are with the Department of Computer Science, University of California Davis, Davis, CA 95616. E-mail: {banerjea, ghosal, mukherje}@cs.ucdavis.edu.*
- *W.-c. Feng is with the Department of Computer Science, Virginia Tech, Blacksburg, VA 24061-0002. E-mail: feng@cs.vt.edu.*

Manuscript received 17 Oct. 2006; revised 7 Mar. 2007; accepted 30 Mar. 2007; published online 9 May 2007.
Recommended for acceptance by M. Singhal.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0367-1106. Digital Object Identifier no. 10.1109/TPDS.2007.1112.

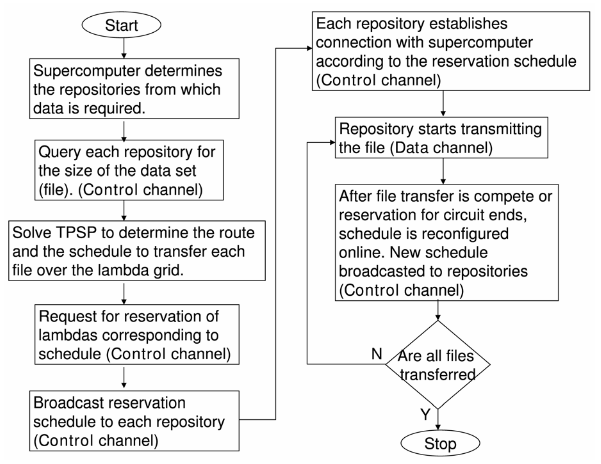
Start
Supercomputer determines the repositories from which data is required.
Query each repository for the size of the data set (file). (Control channel)
Solve TPSP to determine the route and the schedule to transfer each file over the lambda grid.
Request for reservation of lambdas corresponding to schedule (Control channel)
Broadcast reservation schedule to each repository (Control channel)
Each repository establishes connection with supercomputer according to the reservation schedule (Control channel)
Repository starts transmitting the file (Data channel)
After file transfer is compete or reservation for circuit ends, schedule is reconfigured online. New schedule broadcasted to repositories (Control channel)
Are all files transferred
N
Y
Stop

Fig. 1. File transfers from remote repositories to a supercomputer.

problems include single-processor scheduling, multiprocessor scheduling, and open-shop, flow-shop, and job-shop scheduling problems to name a few. However, these are not applicable in our problem setting, since the resources (lambdas) that we consider are not independent; rather, they have connectivity relationships among them. The closest problem setting considers scheduling file transfers over a network when file sizes and the maximum number of file transfers possible from each node are given [11]. This problem considers a fully connected mesh network, and hence, the algorithms described are not applicable to lambda grids, which have sparse connectivity.

Independently, the problem of reserving bandwidth in a lambda grid for a prespecified connectivity has been studied by many researchers. A bandwidth scheduling algorithm that computes the available time slots on a lambda grid between the source and destination has been studied in [20]. The same authors have proposed algorithms for computing the quickest paths, with a minimum end-to-end delay, to transfer a message of a given size from its source to a destination when bandwidth and delay constraints on the links are specified [21]. The Virtual Finish (ViFi) [14] heuristic schedules file transfer over a shared path, depending on the earliest finish time for each file determined from a fair sharing scheme. A Varying-Bandwidth List Scheduling (VBLS) heuristic to compute varying bandwidth levels for different time ranges for a circuit over a lambda grid was studied in [25]. None of the above considers the problem of routing connections.

The focus of this work is on a mathematical model and a greedy approach to solve the problem of integrated routing and scheduling on a lambda grid. The particular example that we consider is of a supercomputer aggregating files from remote repositories. The flowchart describing this process is shown in Fig. 1. We assume the existence of a separate control channel for signaling. The supercomputer determines the repositories that it needs data from and queries for the file sizes. The offline scheduling problem is solved to determine the route and the schedule to transfer the files. The lambdas are reserved for the corresponding schedule of file transfers. During actual file transfers, the schedule may need to be readjusted online to accommodate the actual amount of time that it required to transfer a file. This process is repeated till all files are transferred. In this study, we propose and investigate the characteristics of algorithms to solve the offline scheduling problem and the online reconfiguration problem, as described in the flowchart in Fig. 1.

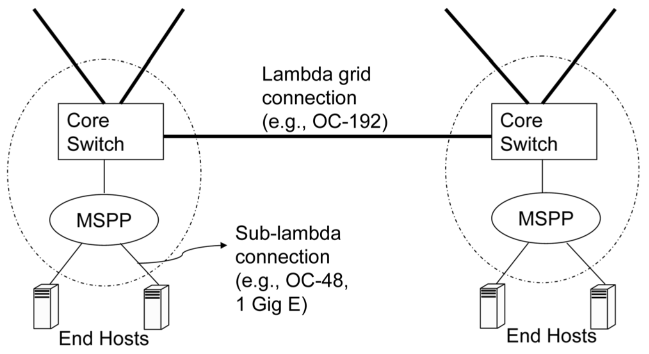
Lambda grid connection (e.g., OC-192)
Core Switch
Core Switch
MSPP
MSPP
Sub-lambda connection (e.g., OC-48, 1 Gig E)
End Hosts
End Hosts

Fig. 2. Illustration of a connection between two end hosts using a lambda grid.

This paper is organized as follows: In Section 2, we model the offline scheduling problem as a Time-Path Scheduling Problem (TPSP) and discuss a corresponding graph-based formulation. In Section 3, we prove TPSP to be NP-complete. We formulate a Mixed Integer Linear Program (MILP)-based mathematical model to solve TPSP. We then discuss a greedy approach based on some heuristics to yield fast and approximate solutions to TPSP in Section 4. Since the actual transfer time for a file may be different from this estimate determined by the offline schedule, we propose an online reconfiguration algorithm in Section 5 for modifying the offline schedule (determined by solving TPSP). We discuss several illustrative examples for both the offline scheduling and online reconfiguration algorithms on sample lambda-grid network topologies in Section 6. We conclude our work in Section 7.

2 Problem Formulation

A lambda-grid network topology, an example of which is the USN [2], may be represented as a graph $G(V, E)$, where each node V represents a core switch, and the edge E represents the connectivity between core switches. Core switches are connected with single or multiple lambdas (a lambda is an optical connection established over a certain wavelength). A core switch is attached to a Multiservice Provisioning Platform (MSPP). MSPPs provide a Synchronous Optical Network (SONET)/Synchronized Digital Hierarchy (SDH) and Ethernet channels at sublambda granularities to end devices such as Storage Area Networks (SANs), data warehouses, or host computers. Thus, a lambda grid may provide an end-to-end connection between two end-host machines via the MSPPs and core switches [20]. The connection from the core switch to the MSPP to the end host is not represented in graph G.

The layout of the end-to-end connectivity is shown in Fig. 2. For example, a simple way by which an end host may connect to a lambda grid is by using a Gigabit Ethernet interface card over a Local Area Network (LAN) connected to the MSPP. Alternatively, it may be connected via a 2.5-Gbps (OC-48) SONET connection. We term this connection from the MSPP to the end host as a sublambda connection. In order to simplify the problem setting, we assume that all end hosts are connected to the MSPPs with the same connection

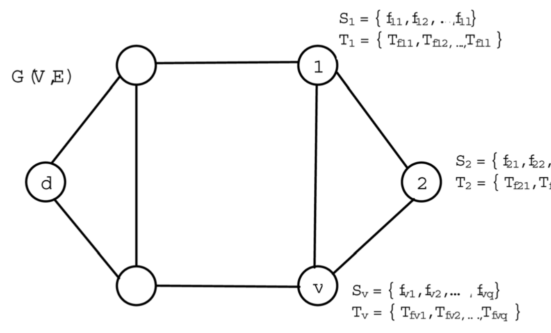
G (V ,E)
S1 = { f11 , f12 , . . ,f1l }
T1 = { Tf11 , Tf12, ...,Tf1l }
S2 = { f21 , f22 , . . ,f2p }
T2 = { Tf21 , Tf22, ...,Tf2p }
Sv = { fv1 , fv2 ,... , fvq }
Tv = { Tfv1 , Tfv2, .. .,Tfvq }
d
1
2
v

Fig. 3. Problem formulation of TPSP on an example six-node network.

bandwidth (that is, 1 or 2.5 Gbps), and therefore, the granularity of each sublambda connection is the same.

A supercomputing machine has high bandwidth connectivity to the MSPP and, thus, has access to all the connections arriving at the MSPP that it is connected to. The node on the graph to which the supercomputer is attached is marked as $d \in V$. At a certain step in the computation, the supercomputer may require data aggregated from multiple end hosts (data warehouses, SANs, etc.) before it resumes computation. We model this process as the transfer of *files* from each end host to the destination supercomputer. All the data that must be transmitted from one end host is modeled as one file. We assume that the connection between the core switch, MSPP, and end host is devoid of congestion and is available at all times. Hence, we do not model this connection in the graph G and mark the core switches that are connected to the end host as the source of the file. Since we assume that all end hosts are connected to the MSPP with the same connection bandwidth, only a single connection of sublambda bandwidth may be established between the end host and the supercomputer at one time.

The mathematical representation is given as follows: At each core switch $v \in V$, there exists a set of files $S_v = \{f_{v1}, f_{v2}, \dots, f_{vl}\}$ corresponding to the end hosts that it is connected to, whose estimated transfer time over the lambda grid to the destination d, T_f is known and is denoted by the set $T_v = \{T_{f_{v1}}, T_{f_{v2}}, \dots, T_{f_{vl}}\}$, where $T_{f_{v1}}$ is the transfer time for file f_{v1}. One way to estimate the transfer time for the respective files is to use *file transfer profiles*, which is briefly discussed in Section 5. An illustration of the problem formulation on a six-node network is shown in Fig. 3. The objective is to determine the following:

1. *Route*. This is the path on the lambda grid, via which a file should be transferred from the source to the destination.
2. *Time schedule*. This is the time at which a connection must be reserved on the lambda grid for the corresponding file. This is important because it may not be possible to transfer all the files simultaneously on the lambda grid due to link-capacity constraints.
3. *Minimum finish time*. The objective is to minimize the total time required to aggregate all the data by using the lambda grid. The last file to reach the destination may be the bottleneck for the supercomputer, since computation cannot be completed unless all the data is aggregated.

We define the above problem as the TPSP [9]. The two dimensions of determining both the path and the time

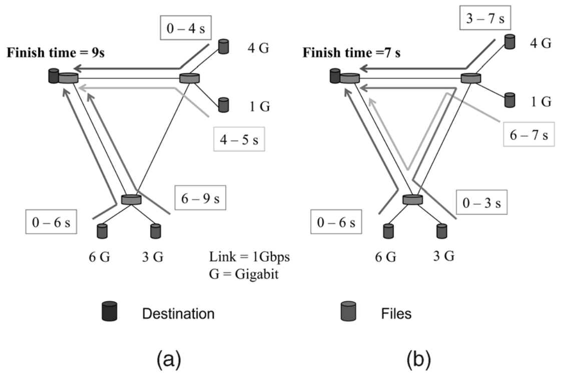

Fig. 4. Example showing that the use of shortest paths may not lead to an optimal finish time.

schedule make this problem exceptionally hard, and it differentiates TPSP from other machine-scheduling problems that have been reported in the literature [19]. We demonstrate this through an example shown in Fig. 4. Four files need to be transferred to a destination node on a three-node lambda grid. Choosing the shortest paths in Fig. 4a leads to a larger *finish time* (9 seconds) than the best possible choice shown in Fig. 4b (7 seconds).

3 Mathematical Model

We prove that TPSP is NP-complete by reducing it to the Multiprocessor Scheduling Problem (MSP) [13].

3.1 NP-Completeness of TPSP

We first model the optimization TPSP as a decision TPSP by asking if TPSP may be solved within a deadline D. Clearly, the problem is in NP because, given a solution, it is easy to verify if the last file reaches the destination within the deadline D and if the constraint that no two files being transmitted along the same path simultaneously is violated. We present a proof for the polynomial reduction of MSP, which is known to be NP-complete, to TPSP.

In the **MSP**, given a set of T tasks, a number $m \in Z^+$ of processors, length $l(t) \in Z^+$ for each task $t \in T$, and a deadline $D \in Z^+$, the goal is to determine if there is a schedule that meets deadline D, given that no two tasks can be processed in the same processor at the same time. The MSP may be reduced to TPSP by constructing the following graph $G(V, E)$, as shown in Fig. 5:

1. Construct a vertex for each processor, resulting in m vertices labeled $1, 2, \dots, m$.
2. Construct a vertex for the destination node denoted by d. Construct one edge from each of the vertices $1, 2, \dots, m$ to d. The weight $w(e)$ of each edge is 1.
3. Construct a dummy vertex for the source denoted by s. Construct an edge from s to each of the vertices $1, 2, \dots, m$. The weight $w(e)$ of each edge is 1.
4. Model all the tasks $t \in T$ as files whose transfer time T_f is the same as the length of the tasks $l(t)$. Node s will be the source node for all these files.

Based on the above definitions, TPSP is formulated as follows: Does there exist a time-path schedule through which the files at node s can be transferred to destination d on graph $G(V, E)$ within time D?

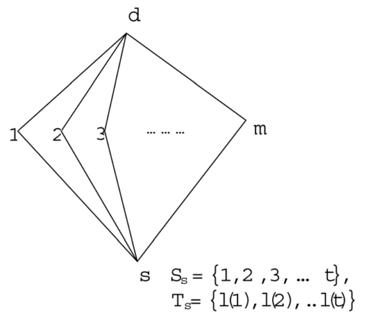
d
1
2
3
m
s
$S_s = \{1, 2, 3, \ldots t\}$,
$T_s = \{l(1), l(2), \ldots l(t)\}$

Fig. 5. Reduction of MSP to TPSP.

We now prove that MSP has a solution if and only if TPSP has a solution. Suppose MSP has a solution. Consider a task t_k which is scheduled at machine p from time τ_k to time $\tau_k + l(t_k)$. This can be scheduled on the path *s*-*p*-*d* from time τ_k to time $\tau_k + l(t_k)$ in TPSP. Since a machine can process only one task at a time, it is guaranteed that the path *s*-*p*-*d* will transfer only one file at a time. Since MSP gives a solution within deadline D, it is guaranteed that TPSP will also have a solution within deadline D. Now, let TPSP have a solution. Then, all files are transferred along one of the paths *s*-*p*-*d*, where $p \in 1, 2, \ldots, m$. Each of these paths may be modeled to one machine in MSP. If the paths of two files share a common link in TPSP, they cannot be scheduled at the same time. This guarantees that one processor is not processing two tasks simultaneously in MSP. Thus, a solution in TPSP has a solution in MSP, and MSP is polynomial-time reducible to TPSP. This proves the NP-completeness of TPSP.

3.2 MILP Formulation

We formulate TPSP as an optimization problem based on the concepts of virtual-topology design in optical networks [18]:

Given:

1. set R of core switches in the lambda-grid network,
2. the core switch at which the supercomputer is located (destination for all files) d,
3. set M of files, which have to be transferred to destination d,
4. physical-connectivity adjacency matrix $P(i,j)$, $\forall i$, $j \in R$, $P(i,j) \in Z^+$ (Here, $P(i,j)$ represents the number of sublambda connections possible between i and j on graph $G(V,E)$, that is, the weight of edge $w(i,j)$. For example, if the wavelength channel capacity is OC-192 (10 Gbps), and the sublambda granularity is OC-48 ($\lambda_{sub} = 2.5\text{Gbps}$), then $P(i,j) = 4$. $P(i,j) = 0$ denotes no connectivity between i and j,
5. the core switch to which the data warehouse at which the files are located, which is connected to (via a LAN or SONET network, and MSPP) $N(m) \in R$, $\forall m \in M$, and
6. estimated transfer rime (T_f) for each file $T_f(m) \in N^+$, $\forall m \in M$.

Subject variables:

1. The virtual-connectivity matrix $V^m_{i,j,k}$ $\forall i$, $j \in R$, $k \in 1 \ldots P(i,j)$, $m \in M$, takes two values: 0 and 1. $V^m_{i,j,k} = 1$ denotes that file m is routed along a path, which contains the link from i to j via sublambda k.
2. Start time $\tau(m)$, $\forall m \in M$, denotes the time at which file m is transmitted. File m is transferred along the route determined from time $\tau(m)$ till time $\tau(m) + T_f(m)$.

Constraints:

1. *Connectivity constraints*. These constraints ensure proper virtual connectivity:

$$\sum_{k=1}^{k=P_{i,j}} V^m_{i,j,k} \le P_{i,j} \quad \forall i, j \in R, m \in M, \tag{1}$$

$$\sum_{j=1}^{j=|R|} \sum_{k=1}^{k=P_{i,j}} V^m_{N(m),j,k} = 1 \quad \forall m \in M, \tag{2}$$

$$\sum_{m=1}^{m=|M|} \sum_{j=1}^{j=|R|} \sum_{k=1}^{k=P_{i,j}} V^m_{j,d,k} = |M|, \tag{3}$$

$$\sum_{j=1}^{j=|R|} \sum_{k=1}^{k=P_{i,j}} V^m_{d,j,k} = 0 \quad \forall m \in M, \tag{4}$$

$$\sum_{j=1}^{j=|R|} \sum_{k=1}^{k=P_{i,j}} V^m_{x,j,k} \le 1 \quad \forall x \in R - d, \forall m \in M, \tag{5}$$

$$\sum_{j=1}^{j=|R|} \sum_{k=1}^{k=P_{i,j}} V^m_{j,x,k} = \sum_{j'=1}^{j'=|R|} \sum_{k=1}^{k=P_{i,j'}} V^m_{x,j',k}, \tag{6}$$

$\forall x \in R - N(m)$, d, $m \in M$.

We use the term *virtual connection* for a sublambda to be determined for transferring a file. Constraint (1) ensures that a virtual connection may not exceed the number of sublambdas available on the physical link. Constraint (2) ensures that an outgoing virtual connection must start from the source node of the file. Constraint (3) ensures that the destination must have one incoming virtual connection for each file. Constraint (4) ensures that the destination must not have any outgoing virtual connection for each file; that is, all connections terminate at the destination. Constraint (5) ensures that there is no bifurcation in the path for a particular file in any node; that is, we consider single-path routing. Constraint (6) is a flow-constraint equation for balanced flows. The number of incoming virtual connections at a node for a particular file should equal the number of outgoing virtual connections for that file for a balanced flow.

2. *No-time-overlap constraints*. These ensure that if a virtual connection exists for transferring one file, then it may be used for another file only after or before the file has been completely transmitted but not during the transmission.

For any virtual connection and pair of files (m, m'), we require one of the following constraints to be satisfied:

$$V^m_{i,j,k} + V^{m'}_{i,j,k} \le 1, \tag{7}$$

$$\tau(m') \geq \tau(m) + T_f(m), \tag{8}$$

$$\tau(m) \geq \tau(m') + T_f(m'). \tag{9}$$

Constraint (7) implies that the virtual connection from node i to node j along sublambda k is not used for transferring both files m and m'. Constraint (8) implies that the virtual connection (i, j, k) is used for transferring file m' only after file m has been transferred. Constraint (9) implies that the virtual connection (i, j) is used for transferring file m only after file m' has been transferred.

3. *Subject variable constraints*. These are given as follows:

$$\tau(m) \geq 0 \quad \forall m \in M. \tag{10}$$

Objective function:

$$Minimize(Max(\tau(m) + T_f(m))) \quad \forall m \in M. \tag{11}$$

The objective function aims at minimizing the time at which the last file is received at the destination, hereafter called the *finish time*.

The *no-time-overlap constraints* and *objective function* can be easily represented as linear equations by introducing some dummy integer variables. Variables $V_{i,j}^m$ and $\tau(m)$ are constrained to be integers. Therefore, the formulation turns out to be a MILP, which can be solved using a commercial MILP solver [4].

The size of the MILP grows exponentially with the number of files because a set of several equations is created for every pair of files. Therefore, in Section 4, we propose a greedy approach based on some heuristics to solve TPSP, and we use the MILP for only a comparative and verification study for smaller topologies and smaller number of files.

Lower Bound (LB) Analysis. Only one file may be delivered to the destination along a sublambda at a time. Therefore, an LB on the *finish time* is given by

$$T_{fin}^{lb} = \frac{\sum_{m=1}^{m=M} T_f(m)}{\sum_{j=1}^{j=|R|} P(j,d)}. \tag{12}$$

It is important to note that the LB on the finish time is based only on the connectivity of the destination node, and it does not consider the complete graph.

4 Greedy Approach for Offline Schedule Computation

We propose a greedy approach for solving TPSP. The greedy approach chooses one file at a time and determines the route along which this file may be scheduled at the earliest. The file is scheduled along this route. We describe two heuristics for choosing the best file and two algorithms for determining the best schedule for a file.

4.1 Heuristics for Choosing the File

4.1.1 Largest File First (LFF)

This approach is based on the intuition that the largest file (having the largest estimated transfer time) is the bottleneck for scheduling because it requires more resources in terms of the amount of time required to be free on the links of the lambda grid. Thus, the largest file remaining to be scheduled is picked as the greedy choice.

```
Algorithm Find_All_Possible_Time_Slots
($source, destination, duration, V$)
  1) Initialize
     $L[i,j] = \begin{cases} \text{Set of available disjoint time intervals} \\ \text{of length greater than } duration \text{ if} \\ i \text{ and } j \text{ are connected,} \\ \text{Null set } \phi \text{ , otherwise} \end{cases}$
  2) $D(s) = \{R^+\}$ , $D(v) = \{\phi\} \quad \forall \quad v \neq s$
  3) for $k = 1, 2, ..., |V|$ do
     for each edge$(u, v)$ do
     $D(v) = D(v) \oplus (D(u) \otimes L[u, v])$
  4) return $D(d)$
```

Fig. 6. Bellman-Ford algorithm to determine all possible time slots ($\oplus$ denotes merging, and $\otimes$ denotes intersection of lists).

4.1.2 Most Distant File First (MDFF)

This approach is based on the intuition that files that are located at nodes far away from the destination in terms of number of hops must be given higher priority for scheduling because they require more links to be free for files to be transferred. Files are chosen in the order of the number of hops that they are located away from the destination.

4.2 Algorithms to Determine Route and Schedule

After a file f is chosen using one of the above heuristics, it may be routed and scheduled on the lambda grid by using one of the following algorithms:

4.2.1 All Possible Time Slots (APT) Algorithm

This algorithm first computes all time slots that are available between the file source $(N(f))$ (denoted as source s) to the destination d for the duration estimated for transferring file $f(T_f)$. We employ the bandwidth scheduling algorithm reported in [22], which is based on the Bellman-Ford shortest path algorithm [12] applied to the disjoint time intervals at which the links are available. The algorithm is described in brief in Fig. 6. If time slots of duration T_f or greater are available before the current *finish time*, then the best fit available time slot is chosen, or else, the earliest available time slot is chosen. File f is scheduled on the chosen time slot and routed along the corresponding path.

The complexity of the algorithm described in Fig. 6 may be written as

$$O(|V| * |E| * (O(\oplus) + O(\otimes)), \tag{13}$$

where $O(\oplus)$ and $O(\otimes)$ are a function of the number of disjoint time intervals on the links ($\oplus$ denotes the operation of merging the disjoint time intervals, and $\otimes$ denotes the operation of intersection of the disjoint time intervals).

4.2.2 K-Randomized Paths (KRP)

This algorithm chooses the best path among K randomly chosen paths. The steps are outlined in Fig. 7. It is important to choose *random* paths because if a fixed set of paths are chosen (for example, K shortest paths), then a few links in the lambda grid may get increasingly congested, and the *finish time* may be poor.

The complexity of step 1 may be stated as $O(K * |V|log|V|)$, since for a sparse graph, we have $(|E| < |V|log|V|)$, and the

```
Algorithm K-Randomized Paths
($source, destination, f, V, K$)
  1) Find random K-alternate paths from the $source$
     to $destination$. Random K-alternate paths may be
     achieved by randomly picking the weights of the links
     and applying Dijkstra's Algorithm to compute shortest
     path [12].
  2) Out of these K paths, choose the one in which file $f$
     may be scheduled at the earliest.
```

Fig. 7. KRP algorithm.

complexity of Djikstra's algorithm is $O(|V|log|V|)$. The complexity of step 2 may be written as $O(K * |V|)$, since $|V|$ is the maximum length of a path in a graph, and we assume that the cost of merging the disjoint time-intervals is a constant. The overall complexity of the algorithm is $O(K * |V|log|V|)$. Typically, the number of alternate paths that needs to be chosen is much less than the number of vertices ($K << |V|$). Therefore, the complexity is

$$O(|V|log|V|). \tag{14}$$

5 Reconfiguring the Schedule Online

The estimated transfer time (T_f) for each file f is required as an input in TPSP. Once an offline schedule is determined using the TPSP solutions based on the estimated file transfer time, the links of the lambda grid are reserved, corresponding to the schedule. When a file is actually transferred in accordance with the offline schedule, two scenarios may occur: either the file is completely transferred within the *circuit holding time*, which is referred to as *early finish*, or it is not fully transferred, which is referred to as *Incomplete File Transfer*. We describe the following algorithms for reconfiguring the schedule online.

Case 1: Early Finish. In case of an early finish, the motivation is to improve the utilization on the reserved links. In particular, when there is an early finish, the present circuit may be torn down as the sublambdas that this circuit was using are now free. There may be future reservations in the offline schedule, which use some of these sublambdas. Since these sublambdas had already been reserved by the current application, the future circuits may be pulled back in time so that the corresponding file transfers can begin earlier than when they were scheduled according to the offline schedule. The algorithm, which is invoked for each file that is transferred early, is presented in Fig. 8.

It should be noted that the above algorithm does not alter the lambda-grid link reservations, which had been made by the offline schedule, but it only alters the start times for the later file transfer. Moreover, if the file transfer start time is modified to earlier than its scheduled time, the end time is kept the same. The *circuit holding time* for that file will therefore increase. This may help us to provide more time margin for a potentially incomplete file transfer event. If this file is transferred before the end time, the online reconfiguration algorithm will attempt to adjust the schedule of the next file transfer.

Case 2: Incomplete File Transfer. The motivation is to handle those cases in which the file could not be transferred in the reserved *circuit holding time*. We assume that the holding time of the current circuit may not be extended, as

```
Algorithm Modify_Schedule_Early_Finish
($Circuit$, $Actual\_Finish\_Time$,
$Scheduled\_Finish\_Time$)
// All three parameters above refer to the file transferred early.
  1) Consider a particular link on the Circuit.
  2) If a file transfer is scheduled to begin at Sched-
     uled_Finish_Time on this link:
       a) Identify other links required for this file.
       b) Check if this file may be scheduled to start
          transfer between $Actual\_Finish\_Time$ and
          $Scheduled\_Finish\_Time$ on these links. If
          $yes$, modify the off-line schedule to start file
          transfer at this time.
  3) Repeat above steps for all links in the Circuit.
```

Fig. 8. Algorithm to modify schedule in case of early finish.

the links may be reserved for transfer of a different file of the same application or for a different application.

For an incomplete file transfer, two different options are available. The first option is to retransmit the entire file after establishing a new circuit. The second option is to transmit only the remaining portion of the file, which could not be transmitted the first time. The former is simple to implement and also does not require any application-level fragmentation and reassembly of file components. However, the time duration in which the file was being originally transmitted is completely lost. The latter requires marking of correctly transmitted sequence numbers by the transport protocol so that retransmission may begin from the last-marked sequence number. Alternatively, checkpointing tools, which are available in many operating systems to maintain persistence of data and recover from failures [26], may be employed. We note that both approaches require establishing a new circuit and hence require new link reservations to be established.

In order to reserve a new circuit, either the APT algorithm or the KRP algorithm may be used.

6 Results

6.1 Offline Scheduling Using TPSP

We chose three different deployed lambda-grid topologies and four other network topologies to demonstrate the performance of the greedy solution to TPSP. We consider the DoE USN [2] superimposed on the NLR network [5]. This is a sparse topology shown in Fig. 9, and the average number of hops between any two nodes is 3.7. Henceforth, we refer to this topology as USN-NLR. The 20-node CANARIE CA*net4 [1] topology is shown in Fig. 10, and its average number of hops between two points is 3.82. A 24-node sample backbone topology of one of the telecom carriers in the US is shown in Fig. 11. The average number of hops between any two nodes in this topology is 2.9. Henceforth, we refer to this topology as 24-NODE. We considered four other symmetric mesh network topologies:

1. the 15-node all-connected topology (ALL-CONNECT),
2. the 15-node bidirectional ring topology (RING),
3. the 24-node (3,2) Shufflenet topology (SHUFFLENET) [18], and

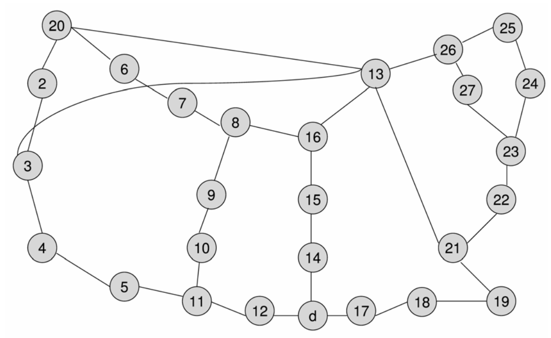
Fig. 9. The 27-node DoE USN superimposed on NLR network (USN-NLR).

4. the 24-node (4,6) Manhattan Street (Torus) network topology (TORUS) [18].

The capacity of each link in all topologies is 10 Gbps (OC-192). We assume that the granularity of each sub-lambda is 2.5 Gbps (OC-48; $\lambda_{sub} = 2.5$ Gbps, and $w(e) = 4$). Since the mesh network topologies are symmetric, any node may be chosen as the destination. We assume that no background reservations exist. The destination in the lambda-grid topologies is denoted as d. A specified number of files of sizes having uniform random distribution between 10 and 20 Gbytes are located randomly across the remaining nodes in the network.

For the offline scheduling, the estimated transfer time for a file m ($T_f(m)$) is determined as

$$T_f(m) = \frac{File\_size}{\lambda_{sub}}. \quad (15)$$

Figs. 12a and 12b show the finish time for a varying number of files transferred on two topologies, USN-NLR and 24-NODE, by using the different heuristics described in Section 4: LFF-APT, MDFF-APT, LFF-KRP, and MDFF-KRP. The value chosen for K in KRP is $K = 5$. (We did not observe any performance improvement beyond $K = 5$.) The LB for the *finish time* calculated using (12) is shown as "LB." To illustrate the importance of the TPSP heuristics, we compare the performance of the heuristics with a simple scheme described earlier in Fig. 4, in which a file is randomly chosen, routed, and scheduled along the shortest path to the destination. These steps are repeated till all the files have been routed and scheduled. We call this scheme RND-SPATH. Fig. 13 compares the finish time for 500 files transferred on the different lambda-grid topologies: USN-NLR, 24-NODE, and CANARIE. Fig. 14 compares the finish time for 500 files transferred on different network topologies: ALL-CONNECT, RING, SHUFFLENET, and TORUS. Fig. 15 compares the finish time for 500 files transferred on lambda-grid topologies with a different value of the sublambda granularity of 1 Gbps. We observe the following:

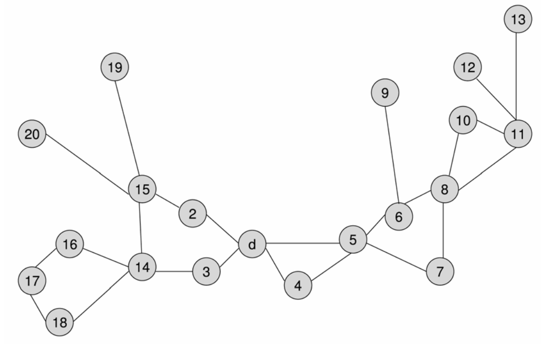
Fig. 10. The 20-node CANARIE (CA*net4).

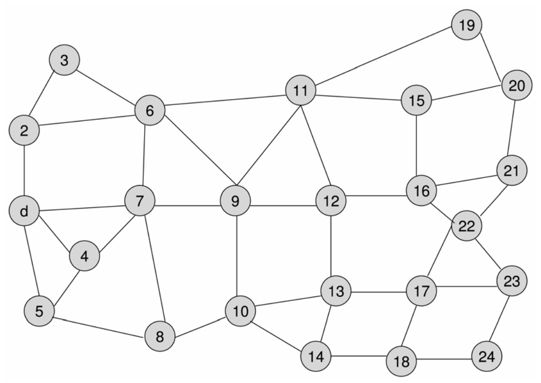
Fig. 11. The 24-node sample backbone topology of a carrier in the US (24-NODE).

1. All the heuristics perform much better than the scheme of routing files on the shortest path to the destination (RND-SPATH). This is because the shortest path becomes increasingly congested as

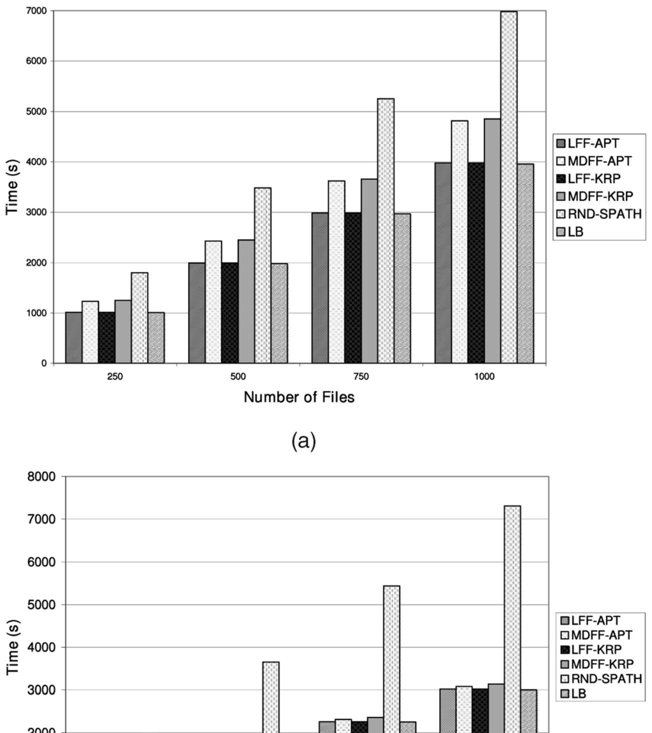

Fig. 12. Comparison of *finish time* for different heuristics with the *LB* for different number of files. (a) USN-NLR. (b) The 24-NODE.

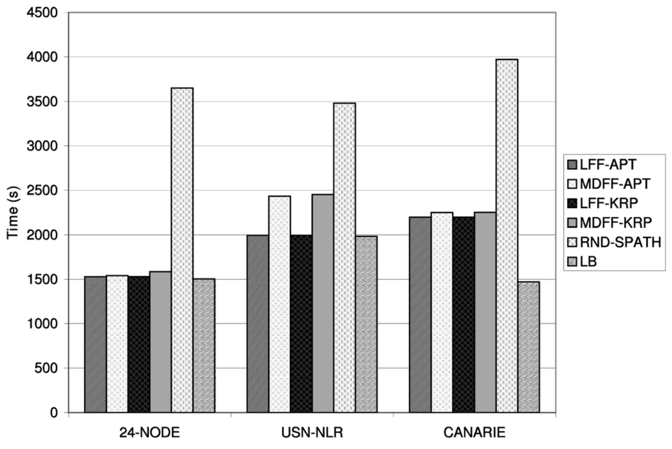

Fig. 13. Comparison of *finish time* for transferring 500 files on different lambda-grid topologies.

more and more files are routed on the same path. This shows the importance of an integrated approach to routing and scheduling, as opposed to treating them separately.

2. The metric of LFF performs better than MDFF on all topologies. MDFF performs poorly on the sparse USN-NLR and CANARIE topologies. This also illustrates the importance of a good heuristic for choosing the files.
3. The LFF-KRP algorithm performs almost as well as the LFF-APT algorithm on the three sparse topologies USN-NLR, CANARIE, and 24-NODE, and on the three network topologies RING, SHUFFLENET, and TORUS. (Note that the APT algorithm is optimal.) Similarly, the MDFF-KRP algorithm performs close to MDFF-APT. However, the APT algorithms perform much better than the KRP algorithms on the fully connected topology because KRP considers only a limited set of random paths, and the number of possible paths in a fully-connected topology is large. Since the complexity of KRP is much less, and lambda-grid networks are usually sparse topologies, we believe that KRP is better suited to lambda grids.
4. LFF-APT and LFF-KRP perform very close to the LB for all topologies, except for CANARIE. The optimal solution is expected to be very close to the LB, except for topologies in which files may be blocked because of lack of links. In CANARIE, links {15-2, 14-3} are a bottleneck for files at nodes {14-20}, and links {6-5, 7-5} are a bottleneck for files at nodes {8-13}. Therefore, the solution yields a transfer time much higher than the LB.
5. Comparing Fig. 13 with Fig. 15, we notice that the file transfer time does not increase noticeably when the sublambda granularity is changed from 2.5 to 1 Gbps. This is primarily because the backbone network and not the edge connection from the MSPP to the end host is the bottleneck for transferring the files.

We compare the performance of the heuristics against solutions for the MILP formulation solved by a commercial MILP solver [4]. Since the MILP does not scale with the problem size, we demonstrate results of the MILP formulation on the small six-node topology in Fig. 3. The capacity of each link is OC-192. The number of files is varied from 15 to 25, and the file size is randomly generated using a uniform random distribution between 5 and 10 Gbytes. Fig. 16 shows the results. All heuristics either match or yield solutions that are very close to the MILP solutions. It is particularly interesting to observe that in two scenarios, MDFF yields the optimal solution. For a general lambda-

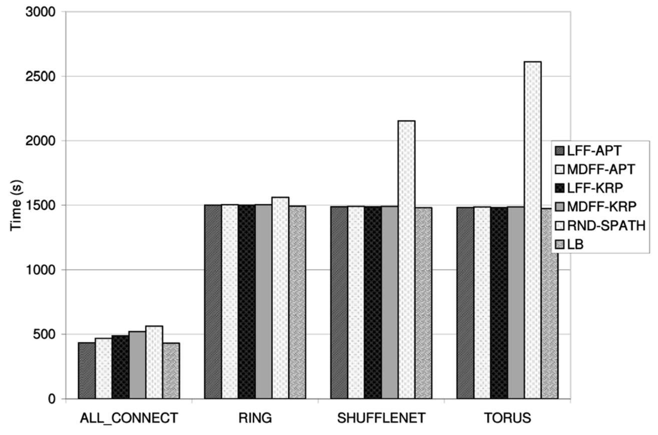

Fig. 14. Comparison of *finish time* for transferring 500 files on different topologies.

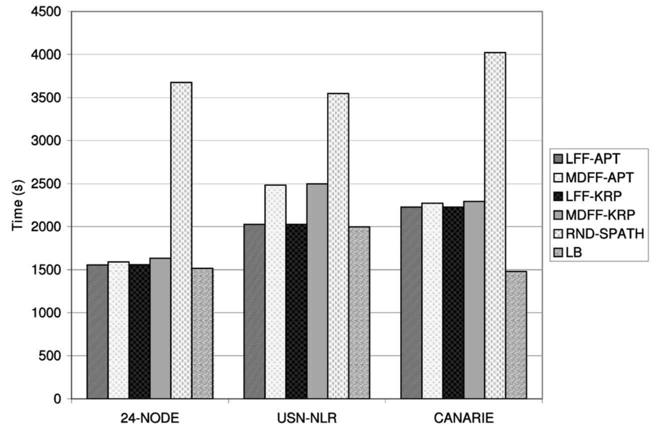

Fig. 15. Comparison of *finish time* for transferring 500 files on different lambda-grid topologies (the sublambda granularity is 1 Gbps).

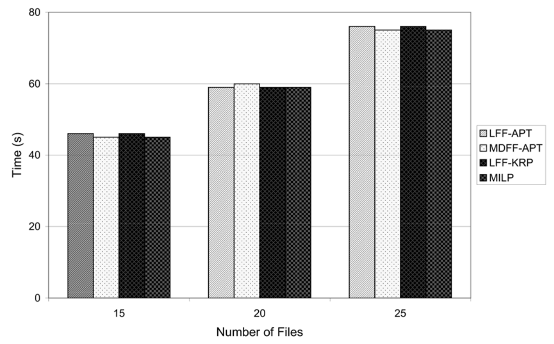

Fig. 16. Comparison of heuristics with the MILP solution for the six-node topology.

grid topology, however, LFF would be the best choice, since MDFF does not perform well in all cases.

6.2 Online Reconfiguration

We emulated file transfers over a lambda grid by transferring files between two machines connected via 1-Gbps Ethernet connection through a dummynet machine (configuration shown in Fig. 17 and Table 1). The purpose of the dummynet machine [23] is to simulate network latency. The dummynet machine receives packets from one host and forwards them to the other after the specified delay. Since we did not have access to high-performance disks, we used Linux RamDisks (which create a temporary file system from the system RAM) to host the file on both machines. Since the system RAM was limited to 1 Gbyte, we experimented with files of sizes between 400 and 800 Mbytes.

Transmission Control Protocol (TCP)-Reno, which has been deployed in the Internet, does not deliver good throughput over networks with a high Bandwidth Delay Product (BDP; for example, the lambda grid which has a high bandwidth and large Round-Trip Time (RTT)). This is because its congestion control algorithm requires a long time to recover from packet loss, however few they may be, thereby decreasing the throughput significantly [15]. Numerous protocols, which are variants of the User Datagram Protocol (UDP), have been developed to deliver higher throughputs in such settings. We used one such protocol, that is, the Reliable Blast UDP (RBUDP) [17], which is available in the QUANTA 1.0 package [7], to transfer the files.

The determination of the transfer time in (15) may not be a perfect estimate. It is nontrivial to accurately estimate the transfer time for a file even though a file of known size is being transferred over a dedicated circuit due to the following reasons:

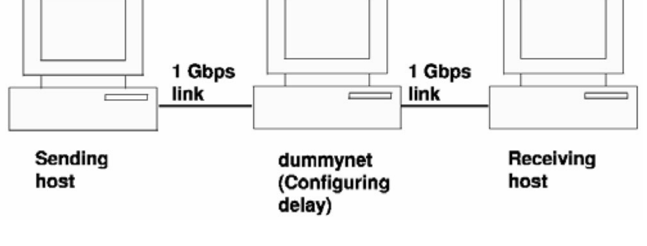

Fig. 17. Example of a dummynet configuration [23].

TABLE 1
Machine Configuration

Processor	Intel Pentium IV, 2.80 GHz for end hosts, Intel Xeon 3.06 GHz with dual PCMCIA bus for dummynet.
Physical Memory	1 GB
Kernel	Linux 2.4.27 for end hosts, Free BSD v4.9 for dummynet
Line card	Intel 1-Gbps Ethernet line cards

1. Often, the end host may not be able to receive data continuously at the high bit rate supported by a dedicated circuit in a lambda grid [27], [10]. As an example, when the end host is under a heavy workload, packets may be lost when the operating system allocates a large context switch to an alternate process. This leads to unpredictable packet losses at the end system. In such cases, the transport-layer protocol may adjust the sending rate if it detects packet losses. This affects the overall time for data transfer.
2. Often, data at the end system is spread across multiple disks in a parallel file system. The load time from the disk may vary from run to run [20].

For the purpose of evaluating our algorithms in this study, we propose to maintain profiles of the end-to-end transfer time for files that are transmitted over the lambda grid. These *file transfer profiles* may be employed to determine the estimated transfer time for a new file. Transfer rates usually vary across file sizes, and file transfer times may not be linearly extrapolated with the file size [24]. Therefore, it is important to maintain *file transfer profiles* for different file sizes. We reiterate that although the above may not be an accurate estimation for the file transfer time, we demonstrate that it performs well in our experimental setting. Some other metrics for the estimation consider the effects of the end host issues highlighted above.

Since the links in a lambda grid must be reserved prior to the file transfers, it is important for the *circuit holding time* (determined from the estimated file transfer time) to be larger than the actual file transfer time so that we do not have to establish another circuit in the future to transfer the same file. The most conservative approach would be to take the largest transfer time (or the lowest transfer bandwidth) out of the past profiles of file transfers to calculate the circuit holding time. Although this would allow almost all files to be transmitted within the circuit holding time, it may lead to poor link utilization. On the other hand, taking a more aggressive estimate such as the mean of past transfer times may lead to a large number of files not being delivered in their allocated times. Therefore, an important problem is how we can accommodate the variance in file transfer times. To accommodate the variance, we consider different numbers of standard deviation (σ) away from the mean (m), which would correspond to the upper limit of a confidence interval in a normal distribution. Such a prediction mechanism is widely deployed in many protocols such as the estimation for the RTT between end hosts in TCP.

We considered the USN-NLR topology, with some modifications. The capacity of each lambda on NLR is 1 Gbps,

TABLE 2
Results for Transfer of 30 Files

Predictor	T_{avg}	T_{max}	N_{max}	T_{min}	N_{min}
m	90.5	200.9	23	130.5	16
$m+\sigma$	91.4	154.6	18	89.7	1
$m+1.25\sigma$	91.6	150.4	16	94.2	5
$m+1.5\sigma$	91.8	151.6	15	90.5	4
$m+1.75\sigma$	92.1	139.0	6	91.2	2
$m+2\sigma$	92.5	92.1	4	90.7	2
$m+2.5\sigma$	93.2	92.5	4	91.2	0
$m+3\sigma$	93.8	93.0	0	92.0	1

m: mean σ: standard deviation.
Predictor: predicted value of the circuit holding time.
T_{avg}: average offline schedule finish time.
T_{max}: maximum observed actual finish time in 10 transfers.
N_{max}: number of incomplete file transfers when T_{max} was measured.
T_{min}: minimum observed actual finish time in 10 transfers.
N_{min}: number of incomplete file transfers when T_{min} was measured.
All times are reported in seconds.

TABLE 3
Results for Transfer of 50 Files

Predictor	T_{avg}	T_{Max}	N_{Max}	T_{Min}	N_{Min}
m	134.8	258.0	44	221.0	34
$m+\sigma$	136.9	204.0	23	137.3	1
$m+1.25\sigma$	137.1	180.2	10	156.5	4
$m+1.5\sigma$	137.7	177.3	13	140.8	2
$m+1.75\sigma$	138.4	167.9	8	135.0	0
$m+2\sigma$	139.4	137.1	2	136.5	0
$m+2.5\sigma$	139.6	137.5	0	136.8	0
$m+3\sigma$	140.8	138.0	0	137.2	0

whereas the capacity on USN (links 3-13, 13-20, and 13-21) is 2 Gbps. The sublambda granularity (λ_{sub}) is 1 Gbps.

The offline schedule is first determined by considering the estimated transfer time using the LFF-KRP heuristic, and the corresponding file transfer events are generated in a Java-based discrete event simulator. For each file transfer event, a file of the same size is transferred between two end hosts via the dummynet. The latency in the dummynet router is set to be the exact end-to-end link latency, which is determined by considering the length of fiber along which the circuit is established in the USN-NLR and considering that the typical delay of communication in an optical fiber is 5 μs per kilometer. Thus, our experimental setup is a close reflection of what the scenario in the lambda grid would be. Once the file has been transferred, the transfer time is measured. The online algorithms mentioned above are invoked thereafter to reconfigure the offline schedule, depending on the actual transfer time.

Our results for the transfer of 30 and 50 files for different predictive schemes for 10 different iterations of transfer are shown in Tables 2 and 3, respectively. Of particular importance are the maximum and minimum *actual finish time* for each predictive scheme.

As expected, using a higher prediction leads to a higher offline schedule finish time. The *actual finish times* are sometimes less than the finish time of the offline schedule, demonstrating the effectiveness of the Modify_Schedule_Early_Finish algorithm. The results show that a limited number of incomplete file transfers do not have an adverse effect on the *actual finish time*. This is because the offline schedule that is generated usually has some links free, and the incomplete files may be transferred using these links. However, if the number of incomplete transfers is high, as it happens when some of the lower predictors are chosen, then the *actual finish time* increases significantly.

Hence, a predictor that limits the number of incomplete transfers to a reasonable number gives a good *actual finish time*. From the above results, we find that the predictors $m+2\sigma, m+2.5\sigma$, and $m+3\sigma$ give the desired values of *actual finish times*. Comparing these three predictors, we find that the highest predictor ($m+3\sigma$) does not lead to the best *actual finish time*. The online *Modify_Schedule_Early_Finish* algorithm tries to pull back circuit start times in case of an early finish. However, since files are sent along different links, if all the links required for the transfer of the next file are not available, the file cannot be scheduled earlier. Hence, long circuit holding times may lead to poor link utilization and may create congestion for another file transfer.

7 Conclusion

In this work, our goal was to present a complete picture of the transfer of large files over a lambda grid for large-scale e-science applications such as GTL. We presented a hybrid approach that combines offline and online scheduling. The TPSP was defined, and a MILP formulation and a greedy approach were proposed to determine the offline schedule. We presented an estimation model for predicting the file transfer time and then proposed an online reconfiguration to the offline schedule, depending on the actual transfer time of a file. Results demonstrate that the LFF0-KRP algorithm performs well. We also demonstrated the importance of the online reconfiguration on an emulated testbed of a lambda grid.

Acknowledgments

The authors gratefully acknowledge Jeremy Archuleta, formerly of Los Alamos National Laboratory and now at Virginia Tech, for his help in setting up the networking infrastructure for their experiments when at the Los Alamos National Laboratory.

References

[1] CANARIE CA*net4, http://www.canarie.ca, 2007.
[2] DoE UltraScience Net Testbed, http://www.csm.ornl.gov/ultranet/, 2007.
[3] "Genomes to Life Requires New Life from Networks," *US Dept. of Energy (DoE) Workshop,* http://www.csm.ornl.gov/ghpn/genome_wk2003.pdf, 2003.
[4] ILOG CPLEX version 9.0, http://www.ilog.com/products/cplex/product/suite.cfm, 2007.
[5] Nat'l LambdaRail Inc. http://www.nlr.net, 2007.
[6] Optiputer, http://www.optiputer.net/, 2007.
[7] QUANTA 1.0, EVL, http://www.evl.uic.edu, 2007.
[8] TeraGrid, http://www.teragrid.org/, 2007.
[9] A. Banerjee et al., "A Time-Path Scheduling Problem (TPSP) for Aggregating Large Data Files from Distributed Databases Using an Optical Burst-Switched Network," *Proc. IEEE Int'l Conf. Comm. (ICC '04),* June 2004.
[10] A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal, "RAPID: An End-System Aware Protocol for Intelligent Data Transfer over Lambda Grids," *Proc. 20th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* Apr. 2006.
[11] E. Coffman, M. Garey, D. Johnson, and S. Lapaugh, "Scheduling File Transfers," *SIAM J. Computing,* vol. 14, no. 3, pp. 744-780, Aug. 1985.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. MIT Press, 2001.

[13] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[14] S. Gorinsky and N.S.V. Rao, "Dedicated Channels as an Optimal Network Support for Effective Transfer of Massive Data," *Proc. IEEE INFOCOM High-Speed Networking Workshop: The Terabits Challenge,* 2006.

[15] Y. Gu, X. Hong, and R.L. Grossman, "Experiences in Design and Implementation of a High Performance Transport Protocol," *Proc. ACM/IEEE Conf. Supercomputing (SC '04),*

[16] N. Hall and C. Sriskandarajah, "A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process," *Operations Research,* vol. 44, no. 3, pp. 510-525, May-June 1996.

[17] E. He, J. Leigh, O. Yu, and T.A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '02),* 2002.

[18] B. Mukherjee, *Optical Communication Networks.* Springer, 2006.

[19] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems,* second ed. Prentice Hall, 2002.

[20] N.S.V. Rao, W.R. Wing, S.M. Carter, and Q. Wu, "UltraScience Net: Network Testbed for Large-Scale Science Applications," *IEEE Comm. Magazine,* vol. 43, no. 11, pp. S12-S17, Nov. 2005.

[21] N.S.V. Rao, W. Grimmell, Y. Bang, and S. Radhakrishnan, "On Algorithms for Quickest Paths under Different Routing Modes," *IEICE Trans. Comm.,* vol. E87-B, no. 4, 2004.

[22] N.S.V. Rao et al., "Control Plane for Advanced Bandwidth Scheduling in Ultra High-Speed Networks," *Proc. IEEE INFOCOM High-Speed Networking Workshop: The Terabits Challenge,* Apr. 2006.

[23] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *ACM Computer Comm. Rev.,* vol. 27, no. 1, pp. 31-41, 1997.

[24] S. Vazhkudai, J. Schopf, and I. Foster, "Predicting the Performance of Wide Area Data Transfers," *Proc. 16th Int'l Symp. Parallel and Distributed Processing (IPDPS '02),* 2002.

[25] M. Veeraraghavan et al., "Scheduling and Transport for File Transfers on High-Speed Optical Circuits," *J. Grid Computing,* vol. 1, no. 4, pp. 395-405, 2003.

[26] Y. Wang et al., "Checkpointing and Its Applications," *Proc. 25th IEEE Int'l Symp. Fault-Tolerant Computing (FTCS '95),* 1995.

[27] R. Wu and A. Chien, "GTP: Group Transport Protocol for Lambda Grids," *Proc. Fourth IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '04),* 2004.

Amitabha Banerjee (S'02) received the BTech degree in electrical engineering from the Indian Institute of Technology Delhi, New Delhi, India, in 2000 and the PhD degree in computer science from the University of California, Davis, in March 2007. He is currently a member of technical staff at Sun Microsystems. His research interests include investigating performance issues in high-speed networks. He is a student member of the IEEE.

Wu-chun Feng received the BS degree in computer engineering and music and the MS degree in computer engineering from Pennsylvania State University in 1988 and 1990, respectively, and the PhD degree in computer science from the University of Illinois, Urbana-Champaign, in 1996. He is an associate professor of computer science and electrical and computer engineering at Virginia Polytechnic Institute and State University (Virginia Tech). His previous professional stints include Los Alamos National Laboratory, the Ohio State University, Purdue University, University of Illinois, Urbana-Champaign, Orion Multisystems, Vosaic, NASA Ames Research Center, and the IBM T.J. Watson Research Center. His research interests include high-performance networking and computing, low-power and power-aware computing, high-speed monitoring and measurement, and bioinformatics. He is a senior member of the IEEE and a member of the ACM.

Dipak Ghosal received the BTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1983, the MS degree in computer science from the Indian Institute of Science, Bangalore, India, in 1985, and the PhD degree in computer science from the University of Louisiana, Lafayette, in 1988. From 1988 to 1990, he was a research associate at the Institute for Advanced Computer Studies, University of Maryland (UMIACS), College Park. From 1990 to 1996, he was a member of technical staff at Bell Communications Research (Bellcore), Red Bank, New Jersey. He is currently with the faculty of the Computer Science Department, University of California, Davis. His research interests are the control and management of high-speed networks, Internet Protocol (IP) telephony, mobile and ad hoc networks, and performance evaluation of computer and communication systems.

Biswanath Mukherjee (S'82-M'87-F'07) received the BTech (Hons) degree from the Indian Institute of Technology, Kharagpur, India, in 1980 and the PhD degree from the University of Washington, Seattle, in June 1987, where he held a General Telephone and Electronics (GTE) Teaching Fellowship and a General Electric Foundation Fellowship. He joined the University of California, Davis, in July 1987, served as the chairman of the Department of Computer Science from September 1997 to June 2000, has been a professor of computer science since July 1995, and currently holds the Child Family Endowed Chair Professorship. To date, he has graduated nearly 25 PhD students, with almost the same number of MS students. Currently, he supervises the research of nearly 20 scholars, mainly PhD students and including visiting research scientists in his laboratory. He serves or has served on the editorial boards of the *IEEE/ACM Transactions on Networking*, *IEEE Network*, *ACM/Baltzer Wireless Information Networks*, *Journal of High-Speed Networks*, *Photonic Network Communications*, *Optical Network Magazine*, and *Optical Switching and Networking*. He served as the editor at large for optical networking and communications of the IEEE Communications Society, as the technical program chair of IEEE INFOCOM 1996, and as the chairman of the Optical Networking Technical Committee (ONTC) of the IEEE Communication Society from 2003 to 2005. He is a member of the board of directors of IPLocks, Inc., a Silicon Valley start-up company. He has consulted for and served on the technical advisory board (TAB) of a number of start-up companies in optical networking. His current TAB appointments include Teknovus, Intelligent Fiber Optic Systems, and LookAhead Decisions Inc. (LDI). His research interests include lightwave networks, network security, and wireless networks. He is the author of *Optical WDM Networks* (Springer, January 2006) and *Optical Communication Networks* (McGraw-Hill, 1997), a book which received the 1997 Honorable Mention in Computer Science from the Association of American Publishers, Inc. He is a fellow of the IEEE. He received the Distinguished Graduate Mentoring Award from the University of California, Davis, (UC Davis) in 2004. Two PhD dissertations (by Dr. Laxman Sahasrabuddhe and Dr. Keyao Zhu), which he supervised, were winners of the 2000 and 2004 UC Davis College of Engineering Distinguished Dissertation Awards. He is a corecipient of the Paper Awards presented at the 1991 and 1994 National Computer Security Conferences.