

Routing and Scheduling Large File Transfers over Lambda Grids

Amitabha Banerjee[†], Wu-chun Feng[‡], Biswanath Mukherjee[†], and Dipak Ghosal^{†1}

[†] University of California, Davis

[‡] University of California, Los Alamos National Laboratory

Abstract— In many application domains, there exists a need to aggregate information from information repositories distributed around the world. In an effort to better link these resources in a unified manner, middleware researchers put forth the notion of a grid. With this context, we consider the problem of aggregating files from distributed databases to a (grid) computing node over a lambda grid. The challenge is to identify concurrent routes (i.e., circuit-switched paths) in the lambda grid network, along which files should be transmitted, and 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. The approach first constructs an offline schedule based on past profiling of transfer rates. Then, as files are being transferred, the schedule is modified online, depending on the amount of time that it actually took to transfer the files. The objective is to minimize the total time required for data aggregation. To demonstrate the effectiveness of our approach, we present experimental results using grid nodes running over an emulated lambda grid topology.

I. INTRODUCTION

Many emerging large scale scientific experiments involve distributed scientific computing. Large quantities of data may be fetched by a computational experiment from multiple information repositories spread geographically. The Genomes to Life (GTL) project [1] and the OptIPuter project [2] are two examples. From time to time a supercomputer may need to aggregate large quantities of data from numerous data warehouses before processing, analysis and visualization. Since data aggregation is at run time, minimizing the time required for data aggregation is the key to improving the overall system throughput.

Many optical circuit switched networks have been provisioned for supporting such large-scale distributed computing. Examples are CANARIE’s CA*net, NetherLight and National LambdaRail [3]. Software is being developed for rapidly provisioning an end-to-end lightpath for dedicated bandwidth (referred to as “User Controlled Lightpath”), as and when needed by an application [4]. Such networks which provide dedicated wavelengths for point-to-point connectivity have been referred to in the literature as lambda grids.

Our objective in this work is to devise a model for such large-scale file transfers, which may be required by GTL and OptIPuter like applications. The problem is formulated as follows. Let $G(V, E)$ denote an optical circuit switched network (lambda grid) topology, where V denotes a geographical end node and E denotes an optical fiber between two nodes. $w(e), e \in E$ denotes the number of wavelengths available on each fiber. Data files are to be transferred from some of

these nodes to a *supercomputer* at node d . Let $f_{i,j}, j \in V$ denote these files, $|i|$ being the number of files at a node. $S(f_{i,j})$ denotes the size of each file. The objective is to find the following:

- 1) Route in the network along which a circuit should be established to transfer each file. The route determines the required *virtual links* (wavelengths) in the lambda grid to establish the circuit.
- 2) The time interval in which the circuit should be established, for the file transfer to take place.

An algorithm for scheduling file transfer on a dedicated optical path, which considered allocation of varying bandwidth levels for different time ranges was considered in [5]. Our work couples routing with scheduling, a challenging task.

The protocol for data aggregation is as follows. The files needed for computations, and their respective sizes are first determined. In the case of high-speed file transfers over dedicated circuits, the bottleneck shifts from the network to the end-host [6]. Therefore, although a file of known size is being transferred over a dedicated circuit, the file transfer time shows some variance because of the end host performance and thereby the flow and rate control by the transport protocol. Hence, we consider predicting the circuit holding times based on previous profiles of file transfer rates. These predictions are used for developing an offline schedule of file transfers. This offline schedule depicts the route and the transfer interval for each file $f_{i,j}$. Let $T_s(f_{i,j})$ and $T_e(f_{i,j})$ denote the start time and the end time respectively for the transfer of each file. *Virtual link* reservations are then made on the lambda grid for this file transfer from node j to node d over the determined route, for the time interval $T_s(f_{i,j}) - T_e(f_{i,j})$.

The actual transfer times are expected to differ from the predicted values. Therefore the file transfer schedule constructed by the offline problem, is modified online as file transfers occur. In this process, we take care not to affect *virtual link* reservations that were already made by the offline schedule, but try to use the freed *virtual links* for future transfers. The primary objective is to minimize the time required for data aggregation, which we define as the *actual finish time*.

It should be noted that offline scheduling is important in order to make the link reservations. Offline scheduling may avoid congested links/ hotspots in the lambda grid, so it is very valuable. The other alternative would be to transfer as many files as possible at a time. The next file may be transferred only after some links are freed due to the complete transfer of a previous file. While this approach may be efficient for a dedicated network, it is not appropriate in a shared network where congestion may arise due to background traffic.

¹This work was supported by a joint UC-LANL CARE grant.

II. OFFLINE SCHEDULING

In our previous work [7], we modelled the offline scheduling problem as a Time-Path Scheduling Problem (TPSP). We showed that this problem is NP-complete, and devised an Integer Linear Programming based mathematical model for an exact solution, and three heuristics to achieve approximate solutions. We showed that the heuristics yielded favorable results in much less time complexity than the integer linear program. Here we consider one of them, the Longest File First (LFF) heuristic for offline scheduling.

LONGEST-FILE-FIRST (LFF) SCHEDULING

This heuristic is based on the intuition that the longest file (having the largest 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, for it to be transferred. Therefore, the LFF algorithm aims at scheduling the longest files first, so that they get priority on the network's resources and get scheduled earlier. For choosing the path over which to transfer a file, the algorithm chooses the best path among K randomly chosen paths. The idea is to prevent one set of links from being selectively congested by files from a node, as would happen if we had always chosen the shortest path.

The steps are outlined below.

- 1) Choose the file F having largest transfer time which has not yet been scheduled.
- 2) Find random K - alternate paths from the source node of file F to destination d . Random K - alternate paths may be achieved by randomly picking the weights of the links and applying Dijkstra's algorithm to compute the shortest path [8].
- 3) Out of these K paths, find one in which file F can be scheduled at the earliest.
- 4) Repeat (1) until all files are scheduled.

The above algorithm assumes that file transfer times are provided to us. However, it is difficult to predict file transfer times accurately because the end-host performance is unpredictable. A prediction framework based on past transfer profiles is proposed in [9]. Various predictors such as weighted average and median based predictors were suggested. Most predictors had average error margins of 6 – 15% compared to measured transfer times. Our experiments using recently proposed protocols for large bandwidth-delay product circuits yielded similar results. We considered two different protocols, UDT [11], and RBUDP [12]. We tuned the protocols for the best possible transfer rates. A large file of 800 MB is transferred between two machines connected via a dummynet [10] shown in Fig. 1. The configuration of the three machines is shown in Table I. Because we didn't have access to high speed disk hardware, we created a Linux RAM-Disk to store the file on both sender and receiver machines. The transfer delay between the two machines is set to be 50ms using the dummynet. The statistics of the 200 file transfers using UDT and RBUDP, and a sample distribution of the transfer times using UDT, are shown in Table II and Fig. 2. The two machines connected back-to-back

TABLE I
MACHINE CONFIGURATION

Processor	Intel Pentium IV, 2.80GHz 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

TABLE II
STATISTICS OF TRANSFER TIME

Statistics	UDT	RBUDP
Max Transfer Time	19.81 sec	34.67 sec
Min Transfer Time	17.31 sec	11.83 sec
Average	17.87 sec	14.47 sec
Standard Deviation	0.3 sec	4.35 sec
Range/ Mean %	13.9 %	157.8 %
St. Dev. / Mean %	1.7 %	30 %

without the dummynet yielded identical throughput results. Therefore, the dummynet is not the performance bottleneck.

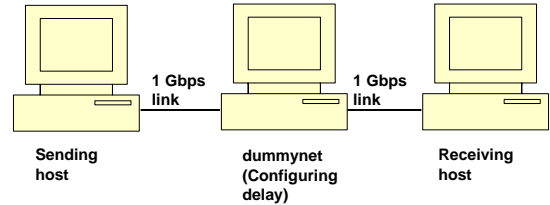


Fig. 1. Dummynet setup [10].

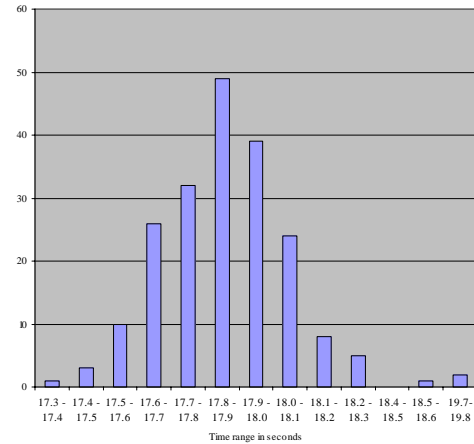


Fig. 2. Distribution of transfer times using UDT.

We observe that the distribution of transfer times is close to a normal distribution. In the case of UDT, although the standard deviation is low (0.3%), the range of observed transfer times is quite high (13.9%). The variations may be explained by the fact that UDT employs flow control and congestion control mechanisms. Congestion control is not significant in dedicated circuits. However flow control is important because at high transfer rates, the receiver buffer at the network line card is frequently overrun, because the multitasking operating system is sometimes not scheduled for transferring data from the buffer to the memory. We observed that as the system load on the receiver is increased, more packets are dropped at

the receiver buffer. We also observed a couple of instances of transfer time being much greater than the mean (in the range 19.7 – 19.8 secs). RBUDP exhibits a much higher standard deviation and range. When the error rate at the receiver buffer is high, RBUDP cannot recover because it doesn't employ flow control. Hence, a large number of retransmits occur.

Since circuits are reserved in advance for file transfers, we would like the circuit holding time to be larger than the actual file transfer time, so that we do not have to establish another circuit in future to transfer the same file. The most conservative approach would be to take the largest transfer time (or lowest transfer bandwidth) out of past profiles of file transfers as the circuit holding time. Although this would guarantee that almost all files are transmitted within the circuit holding time, it may lead to poor circuit utilization. On the other hand, taking a more aggressive estimate like mean of past transfer times would lead to a large number of files not being delivered in their allocated times. The choice of prediction of the circuit holding time is therefore important.

III. ONLINE SCHEDULING

As discussed above, when a file is actually transferred in accordance with the offline schedule, two scenarios may occur. Either the file is fully transferred within the circuit holding time (**Early Finish**), otherwise it is not fully transferred (**Incomplete File Transfer**).

Case I: Early Finish:

In case of early finish, the motivation is to improve circuit utilization on the reserved virtual links. The present circuit may be torn down. The *virtual links* which this circuit was using are now free. There may be future circuits in the offline schedule which use some of these links. Since these links had already been reserved by the current application, the future circuits may be pulled back in time, so that the corresponding file transfers begin earlier than they were scheduled in the offline schedule. The algorithm which is invoked for each file transferred early, is presented below:

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 virtual link on the *Circuit*.
- 2) If a file transfer is scheduled to begin at *Scheduled_Finish_Time* on this link:
 - a) Identify other virtual links 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 offline schedule to start file transfer at this time.
- 3) Repeat above steps for all virtual links in the *Circuit*.

It should be noted, that the above algorithm does not alter the virtual link reservations which had been made by the offline schedule, it only alters the circuit start times. Moreover, if a circuit is indeed modified to start earlier than its scheduled time, the end time is kept the same. Thus the circuit holding

time will increase. This may help providing more margin for an incomplete file transfer event.

Case II: 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 the virtual links may be reserved for transfer of a different file of the same application or for a different application. In case of the former, although the transfer of the next file may be delayed, it delays the entire offline schedule.

For incomplete file transfers, two different options are available. The first is to retransmit the whole file after establishing a new circuit. The second is to transmit only the remaining portion of the file which could not be transmitted. 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. In this work we study the former approach, and plan to discuss the latter if our work is accepted for presentation at the workshop. We note that both approaches require establishing a new circuit and hence require new virtual link reservations to be made.

Algorithm Incomplete_File_Transfer (*File_Number*)

- 1) In the lines of the LFF heuristic, choose K different paths along which this file may be transmitted.
- 2) The predicted transfer time is chosen as the highest transfer time of past transfer profiles. The idea here is to avoid an incomplete transfer again.
- 3) Determine the path along which the file may be scheduled at the earliest. Add this to the offline schedule.

IV. RESULTS

We consider the topology depicted in Fig. 3 for our simulations. This topology is the DoE UltraScienceNet [13] superimposed on the National LambdaRail (NLR) network [3]. Major DoE sites are connected to the nearest network points of presence (PoPs) with 1 Gbps links. The network backbone (shown in bold lines in the figure) may carry 2 1-Gbps connections on two different wavelengths along each physical link. The above are sample numbers for the purposes of illustration, a lambda grid may have a much higher number of wavelengths per physical link. We consider the supercomputer node to be connected to the Houston PoP, and assume that the supercomputer may receive 6 1-Gbps connections simultaneously on different wavelengths, so that there is no bottleneck between the supercomputer and the network. We assume that background traffic is absent, because our objective is to demonstrate the performance of the online algorithms.

Files are of sizes uniformly distributed from 400 MB to 800 MB, and are randomly distributed across the DoE sites. We emulate file transfers on the lambda grid, using the dummynet setup shown in Fig. 1. We limit the maximum file size to

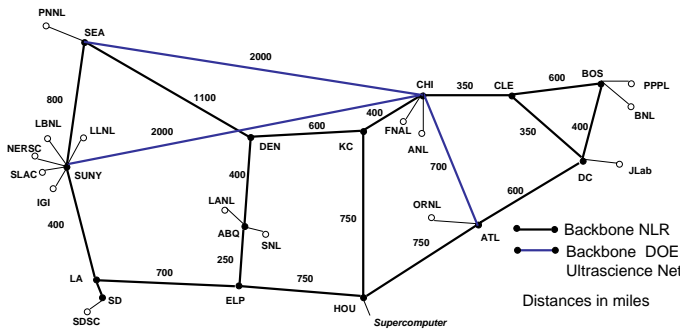


Fig. 3. Topology used for simulation [3], [13]

800 MB because of the limitation of using only RAMDisk-to-RAMDisk transfers. All files are transferred using the network backbone to the *supercomputer*. The transport protocol that we use is UDT [11], tuned appropriately for performance in the bandwidth-delay product settings that we have here.

For prediction of circuit holding times, we maintain the transfer profiles at each transmitting node (DoE site) for file transfers to the *supercomputer*. Since transfer rates may vary across file sizes and file transfer times may not be linearly extrapolated with file size [9], we maintain different profiles at intervals of 100 MB in file size. Within the 100 MB file size range, the prediction of time is linearly extrapolated. Our prediction algorithm considers the past 50 file transfers. We consider different predictors, the predicted value being varied by considering different numbers of standard deviation away from mean, which would correspond to the upper limit of a confidence interval in a normal distribution. Our objective is to predict circuit holding times, which lead to complete file transfers for most files, without leading to poor circuit utilization and compromising on the *actual finish time*.

Once the offline schedule is computed from the predicted values using the LFF heuristic, 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 delay in the dummynet router is set to be the exact end-to-end link delay, as would be if a circuit were established in the network. Thus our experimental setup is a close reflection of what would happen in an actual circuit setup. Once the file has been transferred, the transfer time is measured. The online algorithms mentioned above are invoked thereafter to reconfigure the offline schedule. Our results for the transfer of 30 and 50 files for different predictive schemes are shown in Table III and Table IV, respectively. Of particular importance are the maximum and minimum *actual finish time* for each predictive scheme.

As expected, using a higher prediction, leads to higher offline schedule finish time. The *actual finish times* are sometimes lesser than the finish time of the offline schedule, demonstrating the effectiveness of Algorithm *Modify_Schedule_Early_Finish*. The results show that a limited number of incomplete file transfers does not have an adverse effect on the *actual finish time*. This is because the offline schedule generated usually has some links free, and the

TABLE III
RESULTS FOR TRANSFER OF 30 FILES.

Predictor	$T_{Off_{avg}}$	$T_{Fin_{Max}}$	$N_{R_{Max}}$	$T_{Fin_{Min}}$	$N_{R_{Min}}$
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

TABLE IV
RESULTS FOR TRANSFER OF 50 FILES.

Predictor	$T_{Off_{avg}}$	$T_{Fin_{Max}}$	$N_{R_{Max}}$	$T_{Fin_{Min}}$	$N_{R_{Min}}$
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

m : Mean, σ : Standard Deviation.

Predictor: Predicted value of Circuit Holding Time.

$T_{Off_{avg}}$: Average offline schedule finish time (seconds).

$T_{Fin_{Min}}$: Minimum observed actual finish time in 10 transfers (seconds).

$N_{R_{Min}}$: Number of retransmits for the occurrence of $T_{Fin_{Min}}$.

$T_{Fin_{Max}}$: Maximum observed actual finish time in 10 transfers (seconds).

$N_{R_{Max}}$: Number of retransmits for the occurrence of $T_{Fin_{Max}}$.

incomplete files may be transferred using these links. However, if the number of incomplete transfers are high, as it happens when some of the lower predictors are chosen, then the *actual finish time* increases significantly.

Hence, a predictor which limits the number of incomplete transfers to a reasonable number, gives 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 out of them, 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 early finish. But 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.

V. CONCLUSION

Our aim in this work has been to present a complete picture of transfer of large files over a backbone network for a large-scale scientific computation application like GTL. We present a hybrid approach that combines offline and online scheduling. We then demonstrate the effectiveness of our algorithms by performing a hardware based simulation which emulates an expected real scenario. We envisage that the above hybrid scheduling algorithm would couple with a transport protocol like UDT or GTP [6] for a GTL or OptIPuter like application. Our future work shall focus on a tighter integration in this regard.

REFERENCES

- [1] "Genomes To Life requires new life from Networks," Department of Energy (DoE) (United States) Workshop, 2003, http://www.csm.ornl.gov/ghpn/genome_wk2003.pdf
- [2] OptIPuter project at <http://www.optiputer.net>
- [3] National LambdaRail Inc. at <http://www.nlr.net>
- [4] "CANARIE demonstrates first-of-its-kind software" at http://www.canarie.ca/press/releases/04_06_25.html
- [5] M. Veeraraghavan *et. al.*, "Scheduling and transport for file transfers on High-Speed Optical Circuits", *Proc. PFLDnet 2004, Chicago, USA*
- [6] R. Wu, and A. Chien, "GTP: Group Transport Protocol for Lambda Grids," *Proc. CCGrid, 2004.*
- [7] 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. ICC 2004, Paris, France*
- [8] T. Cormen, C. Leiserson, R. Rivest and C. Stein, "Introduction to Algorithms," Second Edition, MIT Press, 2001.
- [9] S. Vazhkudai, J. Schopf, and I. Foster, "Predicting the Performance of Wide Area Data Transfers," *Proc. of IPDPSI 2002, Fort Lauderdale, Florida, USA*
- [10] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *ACM Computer Communication Review*, 1997.
- [11] Y. Gu and R. Grossman, "UDT: An Application Level Transport Protocol for Grid Computing," *Proc. PFLDnet 2004, Chicago, USA*
- [12] E. He, J. Leigh, O. Yu, T. A. DeFanti, "Reliable Blast UDP : Predictable High Performance Bulk Data Transfer," *Proc. IEEE Cluster Computing*, Chicago, Illinois, 2002.
- [13] DOE UltraScience net testbed at <http://www.csm.ornl.gov/ultranet/>