Utility-Based Scheduling for Bulk Data Transfers between Distributed Computing Facilities

Xin Wang,* Wei Tang,† Rajkumar Kettimuttu,† Zhiling Lan*

*Department of Computer Science, Illinois Institute of Technology
Chicago, IL 60616, USA
xwang149@hawk.iit.edu, lan@iit.edu

†Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439, USA
{wtang, kettimut}@mcs.anl.gov

***Abstract*—Today's scientific applications increasingly involve large amounts of input/output data that must be moved among multiple computing facilities via wide-area networks (WANs). The bandwidth of WANs, however, is growing at a much smaller rate and thus becoming a bottleneck. Moreover, the network bandwidth has not been viewed as a limited resource, and thus coordinated allocation is lacking. Uncoordinated scheduling of competing data transfers over shared network links results in suboptimal system performance and poor user experiences. To address these problems, we propose a data transfer scheduler to coordinate and schedule data transfers between distributed computing facilities over WANs. Specifically, the scheduler prioritizes and allocates resources to data transfer requests based on user-centric utility functions in order to achieve maximum overall user satisfaction. We conducted trace-based simulation and demonstrated that our data transfer scheduling algorithms can considerably improve data transfer performance as well as quantified user satisfaction compared with traditional first-come, first-serve or short-job-first approaches.**

***Keywords*-data transfer scheduling; utility function; simulation**

I. Introduction

Today's scientific applications increasingly involve large amounts of data. The needs for bulk data transfer between remote data centers or computing facilities are growing, such as running data backup on a remote site [18] or moving original data to a remote site for further analysis because of computing resource requirements [28]. Moving the increasing volume of data has imposed a heavy load on the networks between data centers. Especially when multiple data transfers compete for the limited bandwidth, coordinating and scheduling these transfers have become a challenge.

GridFTP (or its software-as-a-service version, Globus [3]) is widely used for bulk data movement. It has been installed on thousands of sites and is being used to move an average of more than 1 PB of data every day. Efforts have been made to improve the performance of a single GridFTP transfer or a set of related GridFTP transfers [23][29]. However, no efforts have been made to coordinate and schedule bulk data transfers as schedulable batch jobs in order to improve the overall experience of all users. As multiple data transfer jobs compete for limited network resources, an unreasonable schedule of data transfers may lead to unbalanced use of the link capability in both temporal and spatial dimensions [27].

To address these problems, we designed and developed a data transfer scheduler to coordinate data transfer requests between distributed computing facilities. Our goal is to *process data transfer requests in an coordinated fashion in order to improve system performance as well as user satisfaction*. Unlike commonly used first-come, first-served (FCFS) or short-job-first (SJF) approaches, our approach quantifies user satisfaction by using a time-utility function (TUF) and schedules data transfer jobs in both temporal and spatial dimensions trying to maximize the aggregate job utility. Here the utility function is a function of job turnaround time and can be used to represent the value (or utility) that the user attaches to the job completion. Maximizing aggregate job utility is consistent with an enhanced overall user satisfaction regarding job turnaround.

We apply our new scheduler to a real case study where data movements are conducted between multiple XSEDE [2] sites using Globus [3]. To evaluate our approach, we developed an open-source event-driven data transfer scheduling simulator named Dsim [1]. Using real job traces from multiple XSEDE sites, we demonstrated that our utility-based data transfer scheduler can achieve enhanced system performance and user satisfaction compared with traditional FCFS and SJF approaches in terms of reduced average job response time, less network contention, and aggregate job utility.

The remainder of this paper is organized as follows. Section II discusses related work; Section III presents the background and the problem statement; Section IV describes our utility-based data transfer scheduling algorithms; Section V introduces Dsim, our event-driven data transfer simulator; Section VI evaluates our new approaches with Dsim using real job traces; and Section VII summarizes our conclusions and briefly discusses future work.

II. Related Work

In the 1980s, one of the earliest works addressing the file transfer scheduling problem [8] proposed list scheduling algorithms whereby transfers are ordered from largest to smallest. Since then, many other algorithms have adapted this idea with the addition of extra parameters [15][11]. The ordering technique usually gives a sub-optimal solution, thus a number of efforts have sought to optimize data transfers via WANs. Khanna et al. [16] divide datasets so that they can be sent over different paths to make full use of network bandwidth. Tummala and Kosar [24] group data transfers on the same path and send them together. Kettimuthu et al. [14] provide an analytical model to optimize concurrency and parallelism. Balman and Kosar [5] suggest a scheduling method that gradually improves the level of concurrency and can achieve a near-optimal value. Arslan et al. [4] propose an algorithm for auto-tuning data transfer parameters such as pipeline, concurrency and parallelism. In practice, Globus [3] serves data transfer requests in FCFS fashion. In our work, we consider the bandwidth as a schedulable resource and develop a scheduler to maximize user-centric metrics in order to achieve better user satisfaction.

The utility function is a concept that is widely used in economics to calculate the relative values of comparable items. Utility functions can be used to represent the value users attach to job completion as a function of time elapsed from when the job was submitted. J. Wang and B. Ravindran [26] propose an efficient packet scheduling algorithm to maximize aggregate utility. Lee and Snavely [9] present precise and realistic utility functions for user-centric performance analysis of schedulers. Vengerov et al. [25] use utility functions for scheduling of data-intensive multiprocessor jobs. Chen [7] proposes utility-based scheduling for multimedia streams in peer-to-peer (p2p) systems. Our work also uses utility functions for data transfer scheduling, which to our best knowledge is the first attempt in this area.

In recent years, network simulation tools have been continuously improved and used for network performance evaluation. According to Jain [12], simulation provides an easy way to predict system performance or compare different approaches. Silvestre et al. [21] propose a p2p simulator that can simulate both middleware-layer and application-layer protocols. ROSS [6] is a massively parallel discrete-event simulation tool that models the operations of the system as a sequence of discrete events in time. Based on ROSS, CODES [10] simulation framework implements multiple network models that can be used to build diverse upper-level application models, such as a data-aware workflow scheduling simulation [22]. In this work, we develop an event-driven simulator that schedules data transfer requests over shared network resources based on the CODES wide-aware network model, which extends the use cases of CODES framework.

III. Background and Problem Statement

In this section, we introduce some concepts of GridFTP data transfers and utility functions as background. We then present the formulation of the problem we will address.

A. Data Transfer with GridFTP

We address data transfer scheduling problems in which GridFTP is used to manage bulk data transfers via WANs. In this context, data transfers are issued by multiple users using GridFTP clients, and data transfers are conducted between GridFTP servers deployed at distributed data centers.

In GridFTP transfers, parallelism and concurrency are two key performance-tuning mechanisms. As illustrated in Figure 1, parallelism involves the use of multiple connections to transfer chunks of a file in parallel from a single-source GridFTP server process to a single-destination GridFTP server process; concurrency involves the use of multiple GridFTP server processes at the source and destination.

In this work, we use both parallelism and concurrency in data transfer. Specifically, we make the TCP connections as sharing and schedulable resources among the queued data transfers.

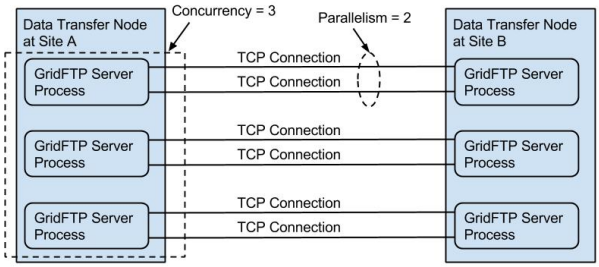
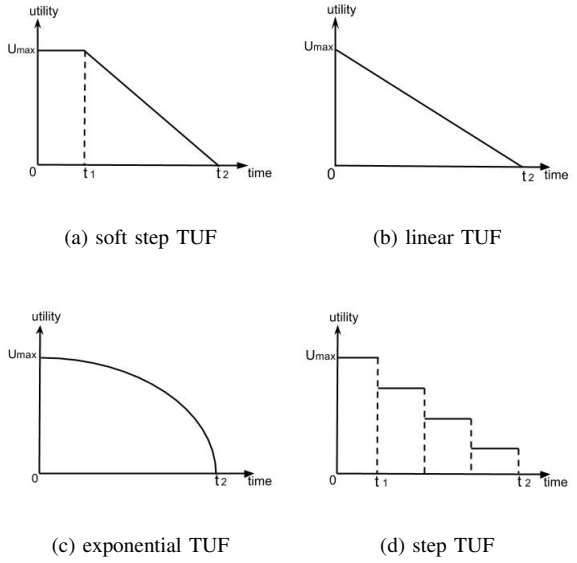


Figure 1. Parallelism and concurrency in GridFTP

B. Utility Functions

Jensen's time-utility functions [13] allow the semantics of soft timing constraints to be precisely specified. A TUF specifies the utility to the system for completing an application or a job. Here, the utility represents the value users associate with their jobs' turnaround time. Figure 2 shows some example TUFs. Usually, the utility function is monotonically nonincreasing; it reaches maximum value at the time when job is submitted and drops to zero at some specified time.

The utility function has recently been applied in batch job scheduling in order to quantify user satisfaction. For example, assuming a job is associated with a utility function U(t) as described in Figure 2(a), the job will achieve (or deliver) a maximum utility value (u_{max}) if it is completed by time t_1, meaning the user gets maximum satisfaction. If the job completes between t_1 and t_2, the delivered utility



(a) soft step TUF (b) linear TUF

(c) exponential TUF (d) step TUF

Figure 2. Different shapes of the time-utility function.

will between the u_{max} and 0, decreasing as time goes by. If the job completes after t_2, the utility is 0, meaning that even if the job is finished, it is too late to deliver any benefit to the user. In this work, we use aggregate utility (the sum of all the utility of completed data transfer jobs) to quantify overall user satisfactions.

C. Problem Statement

Now we describe the data transfer scheduling problem we will address. We target our problem in a distributed environment where each source host has GridFtp service that handles all data transfer requests by moving data to multiple destinations over WANs.

The data transfer requests are modeled as data transfer jobs and are managed in a queue at the source host. Each job include four parameters: the source host where the data is originally locates; the destination host where the data needs to be sent; the size of the data (in bytes); and the submit time of the data request.

For example, in Figure 3 a source host is connected with three destination hosts via a router. A scheduler inside the source host coordinates the data transfers between the source host and the respective destination hosts, via the links with different bandwidths. The decisions made by the scheduler include which jobs should be started now and how many TCP connections should be assigned to each job.

The first aspect involves temporal job sorting/prioritizing. FCFS and SJF are commonly used queuing policies. The former is good for job arrival fairness, and the latter can improve average job turnaround. In addition, other job metrics can be factored into job priorities, such as the "importance"

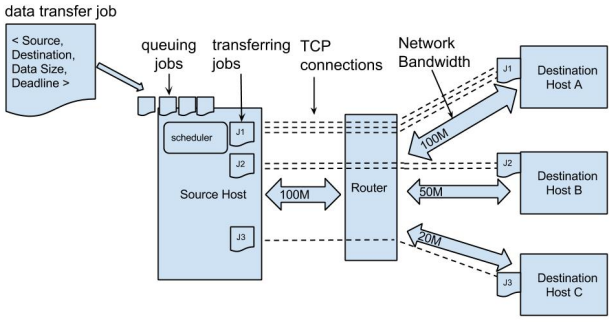


Figure 3. Example of data transfer scheduling problem. A source host is connected with three destination hosts via a router. A scheduler inside the source host will coordinate the data transfers and makes following decisions at each scheduling iteration: (1) which jobs should be started now and (2) how many TCP connections to assign to each job. The scheduling goal is to improve both system performance and user satisfaction.

to user satisfaction, which can be be represented by job utility.

The second aspect involves spatial bandwidth allocation. For example, with limited bandwidth shared, we want to allocate more TCP connections to high-priority jobs so that they can get enhanced transfer. We should also coordinate the TCP connections based on destination bandwidths in order to avoid bandwidth waste. For example, in the figure if we assign an equal number of connections to the three transfer jobs, $J3$ will get up to 33 MB bandwidth between the source host and the router, whereas it can achieve only 10 MB between the router and the destination, resulting in an uncoordinated bandwidth waste.

To generalize the problem, we need a scheduler that coordinates the data transfer jobs by prioritizing jobs in the queue and allocates bandwidths to jobs by assigning TCP connections. The goal of the scheduler is to achieve improved system performance and user satisfaction. Improved system performance can be measured by reduced average job turnaround and network contentions. User satisfaction here can be represented by aggregate job utility.

IV. Methodology

In this section, we describe our methodology in detail. Before discussing the algorithms, we list in Table I the nomenclature used in the text.

Table I
Nomenclature

Symbol	Definition	Symbol	Definition
N	# of destinations	BW	shared bandwidth
Q_i	queue for dest. host i	$size$	job size
F_i	fair share for Q_i	W	window size
M	# of jobs	T_w	job waiting time
C_{max}	max. connections	T_e	estimated transfer time
C_{avail}	available connections	t_c	job complete time
C_j	connections for job j	$U_j(.)$	utility for job j

A. Base Methods

We first describe the base methods that have been used in practice for prioritizing data transfers. In the basic model, the scheduler select one or more jobs with the highest priorities (or at the head of the queue) to start the data transfers. In this work, we examine two policies that will be used as a baseline to be compared with our newly designed scheduling method.

The first policy is first-come, first-served (FCFS), where the scheduler will pick the job with oldest submission time in order to maintain fairness regarding job arrival order. The second policy is short-job-first (SJF), which typically refers to the policy that sorts the jobs based on their lengths (i.e., execution times) to minimize the average waiting time. Here, we examine a modified version of SJF, which considers not only the job lengths but also the job waiting times. That is, the jobs are sorted based on the ratio of waiting time and job size (defined as $T_w/size$). This is to based on the fact that larger (longer) jobs can tolerate longer waiting time than smaller (shorter) jobs.

B. Utility-Based Scheduling

Unlike the base methods, our utility-based scheduling method not only schedules job temporally but also allocates resources (network bandwidth) spatially. In addition, we quantify user satisfaction using the utility function, and we try to maximize the aggregate utility of all jobs.

Specifically, our scheduling algorithm comprises three major steps. First, we try to allocate bandwidths for jobs with different destination with certain fairness guarantees. Second, for the jobs with same destination, we prioritize these jobs and select candidate jobs based on a window size. Third, we apply a utility optimization algorithm to assign TCP connections to selected jobs. Algorithm 1 describes the main algorithm. We will next describe these steps in more details.

Algorithm 1: Utility-Based Data Transfer Scheduling

Input: a set of data transfer jobs J
Output: a subset of jobs J_s that can start, each with an assigned number of C (connections)

```
Q_c = assign_queue_C_min_max_fairness(J, BW);
J_s = [];
foreach q in Q_c do
    J_c = select_candidate_jobs_by_window(q, W);
    J_s^q = sched_C_by_utility(J_c);
    J_s.append(J_s^q);
end
start_transfers(J_s);
```

1) Bandwidth Allocation: For data transfer jobs with different destination hosts, some links are shared by all the jobs, and some are used by jobs with certain destination. For example, in Figure 3 the link between the source and router is shared whereas the links between the router and destinations are separate. For an individual job, the bandwidth of the path from source to destination is bounded by the link with the smallest bandwidth along the path (i.e., the bottleneck link). In this setting the separate links are bottlenecks. Therefore, we want to allocate the shared bandwidth to jobs with different destination based on their respective bottleneck bandwidth.

To allocate the shared bandwidths fairly, we use max-min fairness [17], a technique widely used in practice. The sharing of a network resource is "max-min fair" when the following conditions hold: the lowest demand on data rate is maximized; only after the lowest demand on the network resource has been satisfied will the second-lowest demand on the network resource be maximized; and so on.

Figure 4 shows a simple example. If the bandwidth of bottleneck links are 200, 400, and 500 (MB/s) for each destination, respectively, and the bandwidth of the source link is 1000 MB/s, then the fair share of three destinations will be 20%, 40%, and 40%. Max-min fairness will not assign more bandwidth to a "slow" destination, so that the destinations fully utilize the shared bandwidths.

In this work, we realize the bandwidth allocation by assigning the number of TCP connections to jobs with different destinations, given we have a maximum total number of TCP connections (C_{max}) at the source host. Thus, after the bandwidth allocation step, the jobs in different queues (one queue per destination) are assigned with a certain number of TCP connections, which will be further assigned to each candidate job at later steps.

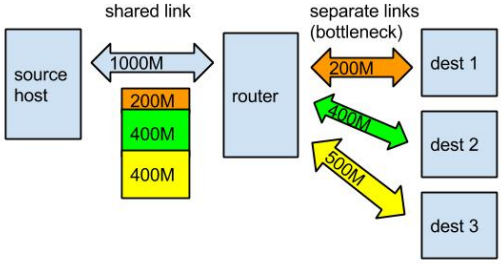


Figure 4. Max-min fairness example. If the bandwidth of separate bottleneck links are 200, 400, and 500 (MB/s) for each destination, respectively, and the bandwidth of the shared link is 1000 MB/s, then the fair share of three destinations will be 20%, 40%, and 40%.

2) Job Prioritizing and Selection: Once we have assigned available TCP connections to each job queue, we then iterate with each of the queues and select a set of candidate jobs that can be started at the current scheduling iteration. That is, the jobs will be sorted with certain policies, and the jobs at the head of the queue will be selected. The sorting criteria can be FCFS, based on the job submission time, or SJF, based on the ratio of job waiting time and the job size ($T_w/size$).

To favor the high-priority jobs, we apply a cap to limit the number of candidate jobs at the head of the queue. We call this cap the window size (W). That is, at most W jobs can be started at the current scheduling iteration.

3) Utility-Based Connection Allocation: Once we select a set of candidate jobs for each queue, we want to decide how many TCP connections for each job to use for data transfers. The more connections a job uses, the more bandwidth it can occupy and the faster the data transfer. Therefore, we can use the connection assignment to favor certain important jobs—jobs that deliver more utility—to achieve aggregate user satisfaction.

In order to assign connections based on utility, the scheduler estimates the utility of a job based on the job's TUF assuming that the job will be started right away. As mentioned earlier, the utility score of a job starts with a maximum value and decays until it drops to 0. We denote the utility function of job i as $U_j(.)$. Thus the utility of job i at time t_c will be $U_j(t_c), i \in [1, M]$, where t_c is the job completion time. For each job queue, the scheduler solves the following problem and finds the optimal TCP connection assignment for each job.

$$\begin{aligned} & max(\sum_{j}^{W} U_j(T_w + T_e)) \\ & s.t. \sum_{j}^{W} C_j = C_{avail} \end{aligned} \tag{1}$$

where C_{avail} is the current available TCP connections, which is no more than the assigned connections in the bandwidth allocation step.

The scheduler selects the optimal assignment of C_j for job j. It needs to know the estimated utility of assigning C_j connections to job j, which is defined as $U_j(T_w + T_e)$. Here T_e is the estimated data transfer time, which is calculated as

$$T_e = \frac{size_j}{f_i * BW} + overhead, \tag{2}$$

where BW is the bandwidth of the shared link, and f_i is the fraction of bandwidth of destination host i, which is calculated by

$$f_i = C_j / (F_i * C_{max}). \tag{3}$$

We implement the utility-based connection assignment using a greedy algorithm. We first initialize a table of estimated utilities by different TCP connections and evenly distribute the total available connections to each job. Next, we conduct connection exchange iterations, at which we reduce one connection from a job and add it to another job such that the reduced utility of the former job is minimal and the increased utility of latter is the maximum. We repeat this step until we cannot increase the aggregate utility any more. The final number of connections will be used as the final connection assignment for the candidate jobs. The job with zero connections will not be started. Figure 5 shows a simple example.

utility / job id / connections	j1	j2	j3	j4
0	**0**	0	0	0
1	**1**	**2**	**2**	**3**
2	2	3	**6**	4
3	3	5	7	5
4	4	6	8	6

Figure 5. Example of the maximum aggregate utility using a greedy algorithm. In the beginning, we assign 4 connections equally to 4 jobs (one per each). After an exchange iteration, we reduce one connection for j1 (utility -1) and add it to j3 (utility +4), because this results in maximum utility increase. In this case, we cannot further increase the utility by the exchange operation. Thus the final assignment is [0, 1, 2, 1] for job j1 to j4.

V. Simulation Framework

Event-driven simulation is an efficient way to understand the system behavior under various workloads, system configurations, and resource scheduling policies. To model the data movement among distributed sites, we developed an open-source simulator named DSim [1]. As shown in Figure 6, Dsim takes inputs such as various job workloads, scheduling policies, and network configurations. With two internal components—queue manager and scheduler—Dsim emulates data transfer scheduling and generates a series of output events, based on which we can analyze the performance metrics. In order to simulate network data transfer, Dsim is built on the CODES network simulation framework [10].

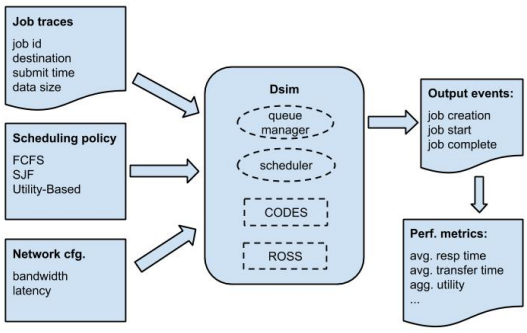


Figure 6. Diagram of Dsim

The goal of the CODES project [10] is to use parallel, fine-grained simulation to explore the design of large-scale storage or network architectures and distributed data-

intensive science facilities. CODES is built on the Rensselaer Optimistic Simulation System (ROSS) [6], a discrete event simulation framework that allows simulations to be run in parallel.

Network simulation is one of key features in the CODES framework. It can be used to model various network topologies. The network congestions between multiple transfers are emulated by breaking the data transfer load into small packets queued up at the ends of the configured network links. Existing work using CODES network models includes the dragonfly network [20] and torus network [19]. Here we use WAN model provided by CODES for data transfer simulation.

VI. Experiments

We evaluate our utility-based scheduling approaches using Dsim. We first discuss the workload characteristics, experiment configurations, and evaluation metrics. We then present our experimental results.

A. Workload Characteristics

We use real job traces collected from at data transfer nodes (DTNs) at four XSEDE sites. The source DTN is associated with Stampede[1], and three different destination DTNs are with Gordon,[2] Mason,[3] and Yellowstone.[4]

The original job trace consists of job IDs, destination host, job submission time, job completion time, and data size. After removing some small-noise jobs, the trace consists of 2,248 data transfer jobs. We categorize these jobs as large jobs and small jobs based on whether they are larger than 1 GB. Table II shows the statistics of the job trace.

Table II
Job Trace Statistics

Category	Job Count	Avg.Size(MB)	Small Job	Large Job
Gordon	797	1068.43	61%	39%
Mason	624	816.38	63%	37%
Yellowstone	897	1032.36	63%	37%
Total	2248	985.20	61%	39%

B. Experimental Configuration

In our experiments, we set up one source host that handles all job transfer requests to three destination hosts via a single router. We configure network bandwidths shown in Figure 4. The shared bandwidth between the source host and the router is set to 100 MB/s, whereas the WAN bandwidth for each destination host is set to 100 MB/s, 50 MB/s, and 20 MB/s, respectively. We conduct our experiments with different numbers of maximum TCP connections varying from 10 to 50. For each configuration of maximum connections, we run simulations with four scheduling policies:

- *FCFS*. Job queue is maintained as a FIFO (first-in, first-out) queue. The scheduler picks the first job in the queue to run until the maximum TCP connections are consumed.
- *FCFS-U*. Job queue is sorted by FIFO, and the utility-based connection allocation is applied after W jobs are selected at the head of the queue.
- *SJF*. Job queue is sorted by the ratio of waiting time and job size ($T_w/size$) . The scheduler picks the first job in the queue to run until the maximum TCP connections are consumed.
- *SJF-U*. Job queue is sorted by SJF. The utility-based connection allocation is applied after W jobs are selected at the head of the queue.

C. Utility Function Setting

In the experiments, we use a soft-step decreasing utility function as shown in Figure 2(a). For each data transfer job, we assign a utility of 100 at the job submission time, which emphasize that each job is equally important. We define the ideal finish time of a job as the time period for transferring the job with one connection. The utility remains at 100 if the job finishes before its ideal finish time; then the utility decays linearly over time. The time point when the utility drops to zero (e.g. t_2 in Figure 2(a)) is different for small jobs and large jobs. We set t_2 to be 20 times the ideal finish time for small jobs, and 40 times the ideal finish time for large jobs. We also set t_2 to be no less than 300 seconds.

D. Evaluation Metrics

In our experiment we used the following metrics:

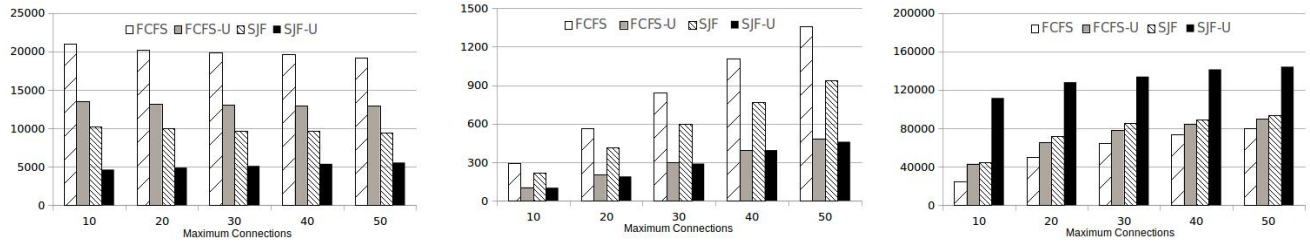
- *Waiting time*: time period between job submission and job start.
- *Response time* (also called turnaround time): time period between job submission time and job completion, consisting of waiting time and data transfer time.
- *Transfer time*: time period between job start to transfer and job transfer time. In this work, the job transfer time of the same job in different test cases is variable because of network contention.
- *Aggregate utility*: As noted in [9], when armed with jobs bearing utility functions, we can directly compute the total value delivered to users:

$$U_{aggr} = \sum_{j}^{M} U_j(t_c)$$

E. Results

We conducted simulations with different scheduling policies and different numbers of maximum TCP connections. Figure 7 illustrates the overall results. The x-axis indicates different value of maximum connections. The y-axis shows

[1]A supercomputer at Texas Advanced Computing Center (TACC)
[2]A computer cluster at San Diego Supercomputer Center (SDSC)
[3]A computer cluster at Indiana University
[4]A supercomputer at Nat'l Center for Atmospheric Research (NCAR)



(a) avg. job response time (sec) (b) avg. transfer time (c) aggregate utility

Figure 7. Performance evaluation for different scheduling policies under various maximum TCP connection configurations. The legends of (a) through (c) represent different scheduling policies. The x-axis is the number of maximum TCP connections.

the average performance results among all the jobs, regarding the metrics described earlier (excluding waiting time as it follows very similar trend with response time).

Figure 7(a) shows the results for average response time. We first observe the utility-based scheduling improves the average response time for the respective queuing policies. That is, FCFS-U reduces the average response time by 34% compared with FCFS; SJF-U improves it by 48% compared with SJF. We also notice that SJF is better than FCFS-U here, meaning that the average response time is more sensitive to the temporal queuing policy (short-job-first) than is the spatial utility-based connection allocation. Using both SJF and the utility-based allocation results in the best performance, a nearly 75% improvement compared with FCFS.

Figure 7(b) shows the results for average transfer time. Although the total job size is the same, the average data transfer time differs in each test case. The reason is that different concurrency levels cause different network contentions. As shown in the figure, using utility-based scheduling improves the data transfer time significantly, no matter what queuing policy is used. Specifically, FCFS-U reduces transfer time 64% compared with FCFS, whereas SJF-U reduces it 52% compared with SJF.

Figure 7(c) shows that using utility-based scheduling can significantly improve aggregate job utility, especially when the queuing policy is SJF. FCFS-U increases aggregate utility from 12% up to 74% compared with FCFS, and SJF-U increases aggregate utility from 54% up to 149% compared with SJF. We also note that the fewer the maximum number of TCP connections, the more significant the utility increases by using utility-based scheduling. This means that the utility optimization algorithm has more impact when a large data transfer workload is sharing limited bandwidth resources.

In addition to the average or aggregate values examined, we present cumulative distributed function (CDF) plots in order to gain more insight into how the performance is improved. We present the CDF plots of one test case to compare the performance variations brought by different scheduling policies and job categories (large or small jobs divided by a 1 GB threshold).

Figure 8 shows the CDF of response times for all jobs. We observe that for small jobs, using utility-based scheduling greatly improves the response time. At the 50th percentile, FCFS-U decreases the response time by 19% compared with FCFS, and SJF-U decreases the response time by 95% comparing to SJF. On the other hand, at the 50th percentile of large jobs, FCFS-U improves response time by 38% compared with FCFS, and SJF-U improves the time 66% compared with SJF. The utility optimization model improves the response time for both small jobs and large jobs.

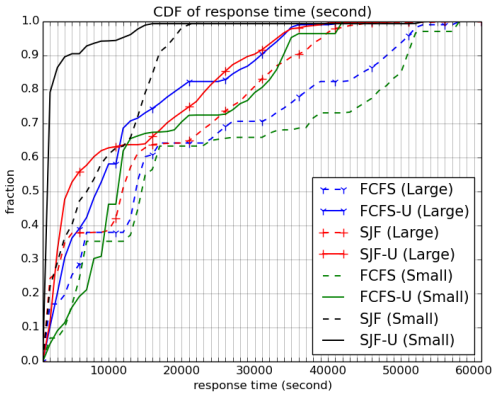


Figure 8. CDF of response time for different policies and job sizes. The y-axis represents the proportion and the x-axis the job response time.

Figure 9 shows the cumulative distributions of data transfer times. We observe that utility-based scheduling improves the transfer time for both small jobs and large jobs. At the 50th percentile of all jobs, FCFS-U decreases the transfer time 64% for small jobs and 43% for large jobs compared with FCFS, and SJF-U increases the time 80% for small jobs and decreases it 60% for large jobs compared with SJF. We can see that about 65% of the small jobs have better transfer times with SJF rather than SJF-U. At the 90th percentile,

however, SJF-U decreases the transfer time 51% compared with SJF for small jobs.

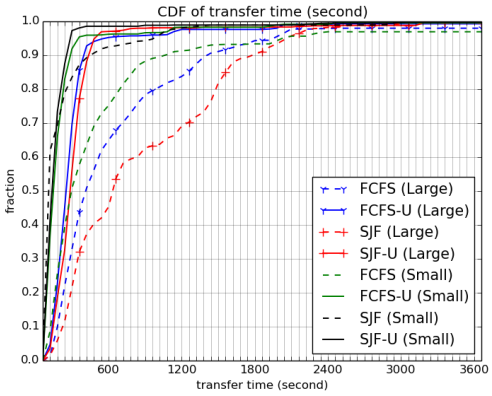


Figure 9. CDF of transfer time for different policies and job sizes. The y-axis represents the proportion and the x-axis the job transfer time.

Figure 10 shows the cumulative distributions of job utility. We first observe that utility-aware policies get better aggregate job utility than does their respective utility-unaware version. That is, the solid lines are all below the dashed lines of the same color. If the queuing policy is SJF, the utility-based algorithm can bring more significant improvement. In terms of job size, we found that large jobs tend to deliver higher utility than do small jobs. That is, the lines with markers are all below the lines without markers. The reason is that the small jobs are more sensitive to the utility decay. Substantial numbers of small jobs have zero utility (we call the proportion of these kinds of jobs "miss rate"). Utility-based algorithms are helpful in this situation. For example, SJF-U decreases the miss rate of small jobs from 78% to 15% compared with SJF.

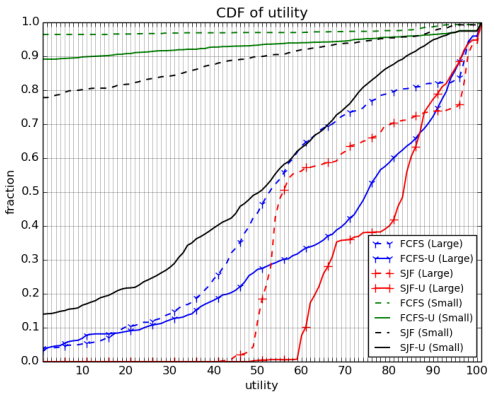


Figure 10. CDF of job utility for different policies and job sizes. The y-axis represents the proportion and the x-axis the job utility.

VII. Conclusion

In this work, we have presented a utility-based data transfer scheduler to coordinate and schedule bulk data transfers among distributed computing facilities via wide-area networks. The goal of the scheduler is to coordinate multiple data transfer requests to improve system performance and overall user satisfactions. We implemented our algorithms in an open-source data transfer scheduling simulator and conducted trace-based simulations using real job traces collected from production computing facilities. The experimental results demonstrate that our approach considerably improves the system performance and user satisfaction compared with traditional scheduling methods FCFS and SJF. Our utility optimization model improves job response time, transfer time and aggregate utility.

We plan to extend our work in following aspects. First, we will improve our approaches to support more diverse workload (e.g. a mixture of batch jobs and real-time jobs with deadlines). Next, we will investigate more sophisticated utility functions for different types of jobs. Finally, we will evaluate our approaches with more complex network topology and dynamic status. That is, we will add more source and destination hosts and introduce unpredictable external load in the simulation.

Acknowledgments

This work is supported in part by National Science Foundation grants CNS-1320125 and CCF-1422009. This material is also based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We give our special thanks to Drs John Jenkins and Misbah Mubarak from Argonne's CODES project team (led by Dr Robert Ross) for their support and consultant in network simulation.

References

[1] Dsim project repo: *https://github.com/xwang149/Dsim*

[2] XSEDE: Extreme Science and Engineering Discovery Environment. https://www.xsede.org

[3] B. Allen, J. Bresnahan, L. Childers, I. Foster, et al., "Software as a service for data scientists," *Commun. ACM*, 55(2):8188, 2012.

[4] E. Arslan, B. Ross, and T. Kosar, "Dynamic protocol tuning algorithms for high performance data transfers," in *Euro-Par 2013 Parallel Processing*, pp. 725-736, 2013

[5] M. Balman, and T. Kosar, "Dynamic adaptation of parallelism level in data transfer scheduling," in *CISIS,* L. Barolli, F. Xhafa, and H.-H. Hsu, eds. IEEE Computer Society, pp. 872-877, 2009.

[6] C. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low memory, modular time warp system," in *Proc. of Workshop on Paral. and Distri. Simulation*, 2000.

[7] F. Chen, "A utility-based approach to scheduling multimedia streams in peer-to-peer systems," in *Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS' 04)*, 2004.

[8] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling file transfers in a distributed network," in *PODC*, 1983.

[9] C. B. Lee, and A. E. Snavely, "Precise and realistic utility functions for user-centric performance analysis of schedulers," in *Proc. of 16th international symposium on High Performance Distributed Computing*, 2007.

[10] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, R. Ross, "CODES: Enabling co-design of multi-layer exascale storage srchitectures," in *Proc. of Workshop on Emerging Supercomputing Technologies*, 2011.

[11] M. Hu, W. Guo, and W. Hu, "Dynamic scheduling algorithms for large file transfer on multi-user optical grid network based on efficiency and fairness," in *ICNS,* J. L. Mauri, V. C. Giner, R. Tomas, T. Serra, and O. Dini, eds. IEEE Computer Society, 2009, pp. 493-498, 2009.

[12] R. Jain, "The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling." Wiley, May 1991.

[13] D. Jensen, "Asynchronous decentralized real-time computer systems," *Real-Time Computing*, 1992.

[14] R. Kettimuthu, G. Vardoyan, G. Agrawal, P. Sadayappan, "Modeling and optimizing large-scale wide-area data transfers," in *Proc. of CCgrid' 14*, 2014.

[15] G. Khanna, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "Scheduling file transfers for data-intensive jobs on heterogeneous clusters," in *Euro-Par, ser. Lecture Notes in Computer Science,* A.-M. Kermarrec, L. Boug, and T. Priol, eds., vol. 4641. Springer, 2007, pp. 214-223, 2007.

[16] G. Khanna, V. Catalyrek, T. M. Kur, R. Kettimuthu, P. Sadayappan, I. T. Foster, and J. H. Saltz, "Using overlays for efficient data transfer over shared wide-area networks," in *IEEE/ACM SuperComputing Conference*, 2008.

[17] A. Ghodsi , M. Zaharia , B. Hindman , A. Konwinski , S. Shenker , and I. Stoica,"Dominant resource fairness: fair allocation of multiple resource types," in *Proc. of NSDI'11*, pp. 323-336, 2011

[18] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Interdatacenter bulk transfers with netstitcher," in *Proc. of the ACM SIGCOMM 2011 Conference*, 2011.

[19] N. Liu and C. D. Carothers, "Modeling billion-node torus networks using massively parallel discrete-event simulation," in *IEEE Workshop on Principles of Advanced and Distributed Simulation*, 2011.

[20] M. Mubarak, C. Carothers, R. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *Proc. of High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion*, 2012.

[21] G. Silvestre and S. Monnet, "Performing accurate simulations for deadline-aware applications," in *MOSPAS*, 2013.

[22] W. Tang, J. Jenkins, F. Meyer, R. Ross, R. Kettimuthu, L. Winkler, X. Yang, T. Lehman, and N. Desai, "Data-aware resource scheduling for multi-cloud workflows: A fine-grained simulation approach," in *Proc. of IEEE International Conference on Cloud Computing Technology and Science*, 2014.

[23] S. Thulasidasan, W. Feng, and M. K. Gardner, "Optimizing GridFtp through dynamic right-sizing." in *Proc. of IEEE International Symposium on High Performance Distributed Computing*, 2003.

[24] S. Tummala, and T. Kosar, "Data management challenges in coastal applications," *Journal of Coastal Research*, special issue no.50, pp. 1188-1193, 2007.

[25] D. Vengerov, L. Mastroleon, D. Murphy, and N. Bambos, "Adaptive data-aware utility-based scheduling in resource-constrained systems," *Research Disclosure*, No. 513, pp. 38-39, 2007.

[26] J. Wang and B. Ravindran, "Time-utility function-driven switched ethernet: packet scheduling algorithm, implementation, and feasibility analysis," in *Proc. of IEEE Transactions on Parallel and Distributed Systems*, 2004.

[27] Y. Wang, S. Su, A. Liu, and Z. Zhang, "Multiple bulk data transfers scheduling among datacenters," *Computer Networks*, 68, pp. 123-137, 2014.

[28] W. Wu, P. DeMar, A. Bobyshev, "An analysis of bulk data movement patterns in large-scale scientific collaborations," *Journal of Physics: Conference Series*, 331(2011), 2011

[29] E. Yildirim and T. Kosar, "End-to-end data-flow parallelism for throughput optimization in high-speed networks," *Journal of Grid Comp.*, 10(3), pp. 395-418, 2012.