



Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: [www.elsevier.com/locate/jpdc](http://www.elsevier.com/locate/jpdc)

# Cluster-to-cluster data transfer with data compression over wide-area networks<sup>☆</sup>



Eun-Sung Jung<sup>\*</sup>, Rajkumar Kettimuthu, Venkatram Vishwanath

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 60439, USA

## HIGHLIGHTS

- We model all the system components involved in end-to-end data transfer as a graph.
- We formulate the problem to optimize data transfer throughput using parallel flows.
- We propose a novel software I/O stack with variable parallel data flows per layer.
- Optimized parallel data flows improve overall data transfer throughput.
- The novel I/O stack with data compression alleviates network bottleneck situations.

## ARTICLE INFO

### Article history:

Received 19 November 2013

Received in revised form

10 May 2014

Accepted 17 September 2014

Available online 28 September 2014

### Keywords:

Disk-to-disk data transfer

System modeling

Optimization

High-speed networks

## ABSTRACT

The recent emergence of ultra high-speed networks up to 100 Gb/s has posed numerous challenges and has led to many investigations on efficient protocols to saturate 100 Gb/s links. However, end-to-end data transfers involve many components, not only protocols, affecting overall transfer performance. These components include disk I/O subsystem, additional computation associated with data streams, and network adapters. For example, achievable bandwidth by TCP may not be implementable if disk I/O or CPU becomes a bottleneck in end-to-end data transfer. In this paper, we first model all the system components involved in end-to-end data transfer as a graph. We then formulate the problem whose goal is to achieve maximum data transfer throughput using parallel data flows. We also propose a variable data flow GridFTP XIO stack to improve data transfer with data compression. Our contributions lie in how to optimize data transfers considering all the system components involved rather than in accurately modeling all the system components involved. Our proposed formulations and solutions are evaluated through experiments on the ESnet 100G testbed and a wide-area cluster-to-cluster testbed. The experimental results on the ESnet 100G testbed show that our approach is several times faster than Globus Online— $8 \times$  faster for datasets with many 10 MB files and  $3\text{--}4 \times$  faster for other datasets of larger size files. The experimental results on the cluster-to-cluster testbed show that our variable data flow approach is up to  $4 \times$  faster than a normal cluster data transfer.

© 2014 Elsevier Inc. All rights reserved.

## 1. Introduction

Scientific workflows are getting more data-intensive as technology advances in sensors, sequencers, detectors, etc. make

abundant data available for analysis. In addition, distributed high-performance computing resources, such as supercomputers, make data movement among geographically distributed sites a major factor that should be taken into account for efficient and reliable scientific workflow management. The increasing amount and complexity of data flows require sophisticated data flow orchestration. Especially, end-to-end data transfers involve many components affecting the overall transfer performance. Disk-to-disk data transfers start with disk reads, go through data processing and data transmission over network, and end up with disk writes. But the process is not simple. For example, disk reads may involve multiple disks on which data are distributed randomly or with some rules.

The recent emergence of high-speed network up to 100 Gb/s has posed considerable challenges and many studies have been

<sup>☆</sup> The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a US Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The US Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

<sup>\*</sup> Corresponding author.

E-mail addresses: [esjung@mcs.anl.gov](mailto:esjung@mcs.anl.gov) (E.-S. Jung), [kettimut@mcs.anl.gov](mailto:kettimut@mcs.anl.gov) (R. Kettimuthu), [venkatv@mcs.anl.gov](mailto:venkatv@mcs.anl.gov) (V. Vishwanath).

conducted on new 100G high-speed networks. In [1], various data transfer middleware such as GridFTP [2] and SRM [3] has been evaluated to determine whether they can saturate a 100G network link. The results in [1] show that they can achieve 80–90 Gb/s in case of memory-to-memory data transfer, where the system's RAM buffer cache is big enough to hold the entire dataset, so the dataset is loaded into memory before data transfer.

Such performance improvements have resulted from several research areas. First, the attempts to optimize network protocols have brought enhanced network throughput. The Globus eXtensible Input/Output System (XIO) [4] provides a framework and API for applications to use different transport protocols without changing the application code. RDMA-based protocols have been evaluated and compared with common protocols such as TCP for high-performance data transfers [5]. The results show that RDMA-based protocols can achieve 10 Gb/s data transfer with much lower operating system overheads and much less host CPU consumption. Another research area focuses on exploiting multiple flows to achieve high-performance data transfer. For example, GridFTP utilizes pipelining [6] and concurrency [7] to offset protocol overhead for small files.

However, because of the lack of a holistic approach to end-to-end data transfer, achieving high-performance data transfer is difficult in varying hardware and software environments. End systems are becoming more and more complex and heterogeneous. System hierarchy is becoming deep and complex with multi-dimensional topologies. Applications must be smart enough to take advantage of parallelism in various sub-systems. So far, manual hardware and software tuning has been needed in order to figure out what configurations are to be set to meet the required data transfer rate. In this paper, we address this problem by modeling system components involved in data transfer and solving mathematically formulated problems.

In this paper, we focus on optimizing parallel flows and CPU loads in end-to-end data transfers. We first show how the throughput for datasets with many files can be improved through optimizing the number of parallel flows under constraints of CPU, disk I/O, and so on. For many applications, the individual file sizes in the dataset are still small with respect to increasing bandwidth-delay products even though the total volume of the datasets has increased significantly in the past decade. For large files, the approach of splitting a file into multiple chunks and transferring the chunks simultaneously improves the performance. However, the same approach does not work with small files—it can even hurt the performance. We show that our approach improves the performance significantly compared with GridFTP and Globus Online, by optimizing parallel data flows.

We propose a new I/O architecture with variable data flows to better improve data transfer throughput. This is motivated by our observation that the same number of data flows at each software I/O layer does not lead to efficient utilization of system resources. For example, the optimal number of data flows from a disk is just one from the view point of throughput whereas the optimal number of data flows for compression is more than one (ideally, it should be equal to the number of cores) to harness the multi-cores in the host to the maximum. In addition, we propose a cluster-wise data transfer algorithm to determine the number of hosts at each cluster such that the data transfer throughput between clusters is maximized when the hardware configuration of the hosts in the same cluster is assumed to be homogeneous.

The remainder of the paper is organized as follows. In Section 2, we describe preliminaries and related work to help understand the context of our work and highlight our contributions. In Section 3, we present graph-based system modeling and data transfer optimization algorithms. In Section 5, we present a novel software architecture for enhanced high-throughput data movement and

cluster-wise data transfer optimization algorithms. In Section 6, we evaluate our approaches in two different testbeds, one of which represents high-speed networks, and the other of which represents a cluster-to-cluster data transfer especially when network bandwidth is a bottleneck. In Section 7, we briefly summarize our work.

## 2. Preliminaries and related work

### 2.1. Globus Toolkit and GridFTP

With the proliferation of grid computing, many grid computing software packages facilitating executing distributed applications on grid resources have been proposed and developed. Among such software packages, the Globus Toolkit [8] is an open source software package for building grids, and GridFTP is a data movement tool in the Globus Toolkit in replacement of ftp, the early well-known file transfer tool. GridFTP has been widely adopted by numerous research institutes for efficient data sharing among collaborative scientists. Globus Online [9] provides a more user-friendly web-based interface and a more reliable and secure file transfers on top of GridFTP.

More specifically, GridFTP has exploited parallelism in many ways for enhanced data transfer throughput with regard to the typical FTP service [10,11]. First, GridFTP provides users with a control parameter for the number of TCP streams. The multiple TCP streams enable an application to fully utilize available network bandwidth because a single TCP stream usually shows saw-shaped utilization due to its AIMD (additive increase multiplicative decrease) property. Another GridFTP feature using parallelism in network protocols is *pipelining* FTP protocol message changes regarding lots of small files (LOSF) transfer [12]. Pipelining allows a client to send many unacknowledged transfer commands at once in order to mitigate the overhead of waiting ACKs for file transfer requests sequentially. In addition, GridFTP allows a client to use concurrent file reads from disks, called *concurrency*, by launching multiple GridFTP servers, which can be regarded as exploitation of data flow parallelism. Overall, we can take two examples, one 10 GB file transfer and many 10 MB file transfer, for the illustration of functions described above. For the former case of one 10 GB file transfer, we may set the number of TCP streams to 3, which leads to three threads sending 3 partitions of a 10 GB file in 3 TCP streams. For the latter case of many 10 MB file transfers, we may set the number of concurrent file reads to 3, which leads to three threads reading and sending different files concurrently. From the perspective of data flows, once the number of data flows can be set by users according to the system configuration and dynamic status such as network congestion, it is usually assumed and implemented that one flow is not split into multiple flows in the course of transferring. For instance, if one process reads data from a disk, the same process usually compresses the data to send over networks. However, if multiple threads for compression are employed, the one data flow from a disk can be split into multiple data flows. Our approach in this paper models and optimizes split and merge of data flows to improve the overall data transfer throughput.

Beyond one-to-one data transfer (one single host to another single host data transfer), m-to-n data transfer (one cluster to another cluster data transfer) is also supported by GridFTP, through striped servers [2]. This m-to-n data transfer was further extended by dynamic data transfer node provisioning in [13].

Along with scalability through striped servers, GridFTP achieves flexibility through an *extensible input/output (XIO) framework* [4]. Basically, the Globus XIO framework provides a user with a dynamically stackable software architecture with regard to simple open/close/read/write (OCRW) file operations as in Fig. 1. Globus XIO framework moves user data to the driver stack, which is dynamically configured by a user, and manages the interactions

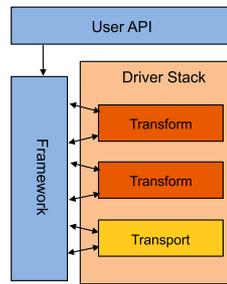


Fig. 1. Globus XIO framework.

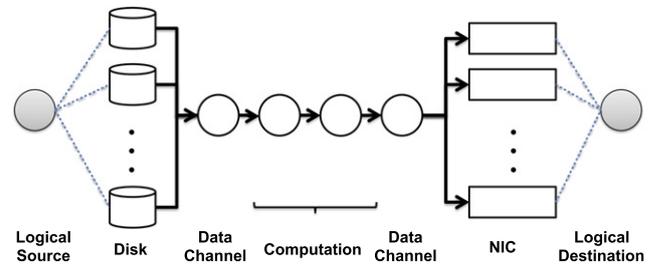


Fig. 2. Data flow graph model.

between stacked drivers. There are two classes of drivers, *transform* and *transport* drivers. *Transform* drivers are doing some data processing such as compression. Some examples of *transform* drivers include *popen* [14] and *compression driver* [4], which perform Unix command and compression operations on data, respectively. *Transport* drivers are located at the bottom of a driver stack and are responsible for sending data outside. Some examples of *transport* drivers [15] include TCP and UDT protocol drivers. Users can write their own drivers along with default drivers supported by the Globus Toolkit. Putting all together, say we want to send files to a remote host via TCP protocol after encrypting and compressing. We can configure the driver stack such that a TCP driver is located at the bottom, and a compression and an encryption driver are placed on top of the TCP driver. The limitation of the current implementation is that all data processing is performed by a single thread. In this paper, we propose a multi-threaded XIO framework to overcome the limitation. Given a multi-threaded XIO framework, the optimized resource allocation is also crucial for maximal data transfer throughput.

## 2.2. Optimizing data transfer

Recently many studies have been conducted on new 100G high-speed networks. Several studies including our work have attempted to optimize the disk transfer throughput in a holistic way by considering all involved system components such as disks and CPUs [16,17]. In [17], the authors proposed heuristics to determine the optimal number of streams and disk/CPU stripes. In our work [16], we formulated the problem as mixed-integer linear programming (MILP) based on a graph model capturing the system topology and characteristics, which is more flexible in terms of system configuration changes and additional computations. However, because split or merge of data flows is not assumed in the system model, we need to extend the previous formulations such that appropriate number of data flows/threads at each driver in XIO driver stack are determined. The new system models and formulations will be presented in detail in the following sections.

## 2.3. Data compression

Data compression has been used in various areas such as multimedia and network domains to improve disk space usage and data transmission time. Recently, motivated by rapid growth of available CPU resources, several studies tried to utilize data compression for I/O throughput in high performance computing systems. In [18], the feasibility of data compression in the I/O forwarding layer was shown through extensive experiments on various compression libraries and datasets in the context of high-performance computing clusters. In [19], efficient data forwarding algorithms using data compression in supercomputers have been proposed. In [20], a framework incorporating various compression and decompression as well as customized compression algorithms for scientific datasets was presented. Data compression is useful

in situations where network bandwidth is a major bottleneck and CPU resources are relatively abundant in the case of one-to-one data transfer or when many applications or hosts are contending for network resources as in the case of m-to-n data transfer. We can think of this problem as balancing allocations of network and CPU resources.

Our work focuses on optimizing the throughput of end-to-end data transfers in wide-area networks using data compression. Our work differs from the previous work in that we propose the layered data processing framework with different number of data flows per layer and algorithms for determining proper numbers of data flows in layers.

## 3. Optimizing single-node disk-to-disk data transfer

In this section, we describe how to model system components relevant to end-to-end data transfer, and we formulate the problem mathematically based on models. We will call this the single-node data transfer (SNDT) problem for the rest of the paper.

### 3.1. System modeling

In this section, we discuss how we can model each component of the system so that we can develop optimization formulations to solve.

The overall system can be modeled as a graph as shown in Fig. 2. In the graph, there are five classes of nodes, and edges that link adjacent nodes. The five classes of nodes are disk node, data channel, computation node, NIC, and logical node. A node is not associated with any attribute, but an edge is associated with attributes describing a node's characteristics. Data channel nodes reflect contention among data flows. For example, if all disks are connected to only one disk interface adapter, maximum disk throughput may not scale linearly as the number of disks increases due to data contention. Logical nodes are inserted for explicit data flow start and end in a graph model. The CPU cores are not expressed explicitly as a node but are put implicitly as costs on edges and constraints in the resulting formulations.

Two attributes are assigned on an edge. One is *capacity/bandwidth* of a source node. The other is *cost* of a data flow on the edge. Both attributes can be either a constant value or a function of some parameters originating from underlying system behaviors. Depending on two end nodes linked by an edge, the edge has different attributes. First, the edge linking from a logical start node to a disk node, *logical edge*, is a logical link with unlimited bandwidth and zero cost function. Second, the edge linking from a disk or data channel node to any other node, *disk edge*, represents a disk I/O path from a disk or data channel. Third, the edge linking from a computation node to another computation node or a NIC node, *compute edge*, represents a data flow going through computations such as GridFTP and compression computation. Fourth, the edge linking from a NIC node to a logical end node, *network*

edge, represents a network path from a source node to a destination node. Each edge is associated with a bandwidth function and a cost function. A bandwidth function and a cost function of an edge describe the performance throughput and CPU resource consumption of a source node, respectively. In the following subsections, we describe each edge's attributes and associated modeling in detail.

### 3.1.1. Disk modeling

A disk edge is associated with disk capacity/bandwidth and CPU load related to disk I/O operations. Even though many parameters such as disk cache size are involved in disk I/O bandwidth, the number of data flows per disk is the most important variable assuming that other parameters are fixed and not adjustable.

Eq. (1) computes the utilization of a disk as a function of the number of processes and disk access probability [21]. Here  $p$  is a ratio of request data size and the stripe size of a RAID disk. If we assume that file size or request data is bigger than the stripe size of a disk,  $p$  can be substituted by 1. The resulting equation is  $U \simeq 1/(1 + \frac{\gamma_d}{L})$ , which means the disk utilization increases to some extent as the number of processes increases.  $\gamma_d$  is a constant to take into account other factors in disk performance such as block size and disk cache.

$$U \simeq 1 / \left( 1 + \frac{1}{L} \left( \frac{1}{p} - 1 + \gamma_d \right) \right) \quad (1)$$

$U$ : Utilization  
 $L$ : Number of processes issuing requests  
 $p$ : Probability that a request will access a given disk  
 $\gamma_d$ : Empirically calibrated value

The disk throughput can be determined by Eq. (2) in which the disk utilization in Eq. (1) is multiplied by  $\frac{N \cdot SU}{E(S)}$ . The equation can be rearranged as Eq. (3) after substituting  $\frac{N \cdot SU}{E(S)}$  by  $\frac{\alpha_d}{L}$ , where  $\alpha_d = N \cdot SU$ , since  $E(S)$ , the expected service time of a given disk request, is proportional to  $L$ . We can determine  $\alpha_d$  and  $\gamma_d$  in Eq. (3) through experimental values. The cost function associated with a disk edge represents the CPU usage for disk I/O from a disk to kernel memory. Since the disk I/O operation is done asynchronously through a hardware interrupt and CPU is not involved at all, we set the cost function for a disk edge to zero. Additional operations such as memory copies for actual reads by an application are accounted by computation modeling in the next section.

$$T = \frac{U \cdot N \cdot SU}{E(S)} \quad (2)$$

$T$ : Throughput  
 $U$ : Utilization  
 $N$ : Number of disks in a RAID disk  
 $SU$ : Stripe size  
 $S$ : Service time of a given disk request

$$T = \frac{1}{1 + \frac{\gamma_d}{L}} \cdot \frac{\alpha_d}{L} = \frac{\alpha_d}{L + \gamma_d} \quad (3)$$

$\alpha_d, \gamma_d$ : Empirically calibrated value

If the source node is a data channel node, the disk edge can be associated with this bandwidth function when the data channel node has fan-in disk nodes, or the disk edge can be associated with infinite bandwidth when the data channel node has fan-out nodes.

Eq. (3) will be used as the bandwidth function  $B_{lk}^n$  in Section 3.2 and approximated by a linear/quadratic function for linear programming solver such as cplex [22].

### 3.1.2. Computation modeling

A compute edge is the edge whose source node is a computation node, and it has attributes of linear bandwidth and cost functions.

The bandwidth function is a function of the number of flows as in Eq. (4), and the cost function can be defined as in Eq. (5).

$$T = \alpha_c n_s + \gamma_c \quad (4)$$

$n_s$ : Number of parallel data transfer streams  
 $\alpha_c, \gamma_c$ : Empirically calibrated value

$$C = \beta_c r \quad (5)$$

$C$ : CPU load  
 $r$ : Data flow rate  
 $\beta_c$ : Empirically calibrated value

Eqs. (4) and (5) will be used as bandwidth function  $B_{lk}^c()$  and cost function  $C_{lk}^c()$ , respectively, in Section 3.2.

### 3.1.3. Network modeling

A network edge is the edge linking a NIC node and a logical destination node, and it has attributes of a throughput function and a cost function. In order to simplify the problem, only TCP is considered and a NIC is assumed to have a preassigned protocol property associated with corresponding throughput function.

Several throughput models for parallel TCP streams have been proposed to predict the performance. The simplest model is proposed in [23] and given by Eq. (6).

$$T \leq \min \left\{ NC, \frac{MSS \times c}{RTT} \cdot \frac{n_t}{\sqrt{p}} \right\} \quad (6)$$

$T$ : Achievable throughput  
 $NC$ : Capacity of NIC  
 $MSS$ : Maximum segment size  
 $RTT$ : Round trip time  
 $p$ : Packet loss rate  
 $n_t$ : Number of parallel data transfer streams

Since  $\frac{MSS \times c}{RTT} \cdot \frac{1}{\sqrt{p}}$  is a constant, Eq. (6) can be rearranged as  $\alpha \cdot n_t$  where  $\alpha = \frac{MSS \times c}{RTT} \cdot \frac{1}{\sqrt{p}}$ .

The cost function for TCP is given by Eq. (7).

$$C = \beta_n r \quad (7)$$

$C$ : CPU load  
 $r$ : Data flow rate  
 $\beta_n$ : Empirically calibrated value

Eq. (6) will be used as bandwidth function  $B_{lk}^n$ , and Eq. (7) will be used as cost function  $C_{lk}^p$  in Section 3.2.

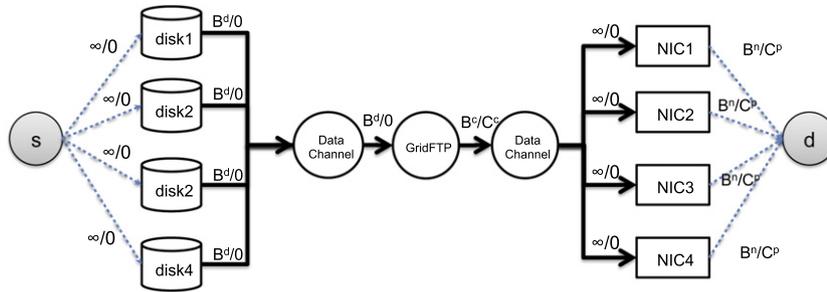
## 3.2. Problem formulation

In order to simplify the problem, the following assumptions are made:

- There is only one machine at each end; Cluster-level modeling and formulation are discussed in the next section.
- There is a dedicated network path between the sender and the receiver machine.
- The data rates of all parallel data flows are same. This means that the total data rate (and data I/O load) is evenly distributed over current parallel flows. Even though disks attached to the machine may have slightly different capacities, we assume homogeneous disk resources in this paper.
- A sender and a receiver have similar hardware such that optimization on the sender side is sufficient for end-to-end data transfer optimization.
- The number of parallel transport protocol (TCP/RoCE) flows can be greater than the number of parallel data flows from disks.

**Table 1**  
Notations for problem formulation.

Notation	Description
$V_s$	Logical source node
$V_d$	Logical destination node
$N_d$	Number of disks
$N_c$	Number of CPU cores
$n_s$	Number of data streams per each disk; integer variable
$n_{lk}^r$	Number of parallel TCP streams on an edge $(l, k)$
$r_{lk}$	Data rate on an edge $(l, k)$
$B_{lk}^d(n_s)$	Disk capacity/bandwidth of $V_l$ , a disk node, associated with an edge $(l, k)$
$B_{lk}^c(n_s)$	Computation capacity of $V_l$ , a computation node, associated with an edge $(l, k)$
$B_{lk}^n(n_t)$	Maximum network capacity/bandwidth of $V_l$ , a NIC node, associated with an edge $(l, k)$
$C_{lk}^c(r_{lk})$	CPU/Computation cost of $V_l$ , a computation node, associated with an edge $(l, k)$
$C_{lk}^p(r_{lk})$	CPU/Computation cost related to network protocol on $V_l$ , a NIC node, associated with an edge $(l, k)$



**Fig. 3.** NERSC host graph model.

#### Objective

maximize  $T$  (8)

#### Subject to:

$$r_{lk} \geq 0, (l, k) \in E \quad (9)$$

$$0 \leq n_s \leq M_s, M_s \text{ is maximum number of data streams.} \quad (10)$$

$$r_{lk} \leq \begin{cases} B_{lk}^d(n_s), & (l, k) \in E, \text{ if } V_l \text{ is a disk node} \\ B_{lk}^c(n_s), & (l, k) \in E, \text{ if } V_l \text{ is a computation node} \\ B_{lk}^n(n_t), & (l, k) \in E, \text{ if } V_l \text{ is a NIC node} \end{cases} \quad (11)$$

$$\sum_{k:(l,k) \in E} r_{lk} - \sum_{k:(k,l) \in E} r_{kl} = 0, \quad l \neq s_j, l \neq d_j \quad (12)$$

$$\sum_{k:(s,k) \in E} r_{sk} - \sum_{k:(k,s) \in E} r_{ks} \geq T \quad (13)$$

$$\sum_{k:(k,d) \in E} r_{ks} - \sum_{k:(d,k) \in E} r_{dk} \geq T \quad (14)$$

$$\sum_{k:(l,k) \in E} C(r_{lk}) \leq \min(N_c \times 100, n_s \times N_d \times 100) \quad (15)$$

**Fig. 4.** Flow optimization algorithm for single node data transfer (SNDT) problem.

The last assumption means that the system can automatically adjust the number of network transfer streams if one network transfer stream is not enough, in order to accommodate the output data rate from computation nodes. For example, GridFTP [2] use multiple logical TCP flows, called parallelism, per one data stream to overcome the limitation of TCP protocol in high-bandwidth high-latency networks.

The overall problem-solving procedure is as follows.

- Compute parameters of capacity functions based on empirical data.
- Formulate the modified multicommodity flow problem based on the capacity/cost functions on edges.
- Find a solution including the number of parallel flows, the number of required CPUs, and the number of NICs using linear programming solver, cplex [22].

- Determine the number of parallel TCP/RoCE flows based on the amount of flows on network edges.

The graph model as in Fig. 3 can be formally represented by a graph  $G = (V, E)$ , where  $V$  is a set of vertices, and  $E$  is a set of edges.

Table 1 gives a list of notations for mathematical formulations, and the complete formulation is described in Fig. 4. The formulation in Fig. 4 is mixed-integer convex programming (MICP) since  $n_s$ , the number of data streams, is integer and the bandwidth function  $B_{lk}^d$  is approximated by quadratic functions.

The objective function is given in Expression (8), which is to maximize overall throughput of data transfer. Expression (10) sets the range of the number of data streams per disk. Expression (11) is a bandwidth/capacity constraint on the edges, where  $r_{lk}$  denotes data rate on an edge  $(l, k)$  and bandwidth functions are chosen depending on the edge type. The flow conservation constraint given by Eq. (12) ensures that the sum of incoming data rates should be same as that of outgoing data rates at every node. In special cases such as compression computation, the sum of outgoing data rates can be a fraction of that of incoming data rates. Expressions (13) and (14) constrain the total outgoing data rates from the logical source and to the logical destination node to be greater than or equal to  $T$ , which is to be maximized. In this way, we can get the solution that maximizes the overall data throughput. The computation constraints by the number of CPU cores in the system and the number of data flows is given by Expression (15). Note that the formulation assumes a circumstance where the number of data flows per disk is the same, but the formulation can be easily extended to reflect different numbers of data flows per disk by assigning separate variables per disk.

#### 4. Enhancing performance with data compression

In this section, we present a novel software architecture based on the Globus XIO framework for enhanced data transfer throughput and modified formulations in accordance with the new software architecture.

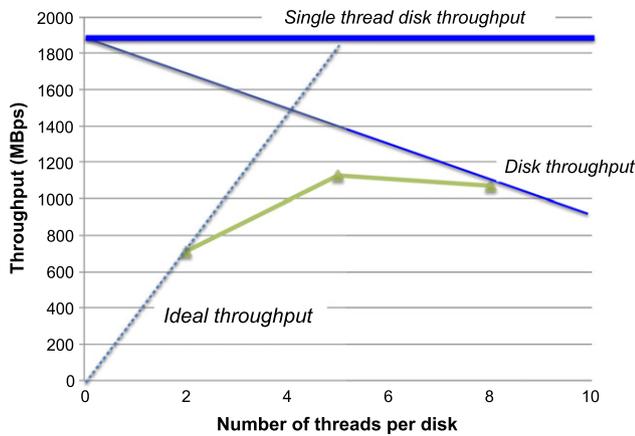


Fig. 5. GridFTP throughput when using *popen* with ‘tar cfz’ option.

#### 4.1. Motivation

In Section 3, we described model-based optimization algorithms for disk-to-disk data flows over 100 Gb/s wide-area networks. We assumed that a certain data flow is not split in the course of data transfer in our graph model shown in Fig. 2. Accordingly, all the bandwidth function of edges in our graph model are either constants, i.e., infinity, or functions of the number of data streams per each disk. The assumption regarding our data flow model is based on the actual implementation of GridFTP [2]. The Globus XIO framework deployed by GridFTP, which is described in Section 2.1, is consistent with our graph model with unsplitable data flows to some extents. Our graph model in particular matches well with the Globus XIO framework since one transform driver in an XIO driver stack can be mapped to one computation node in the graph model.

One noticeable observation from experiments is that a variable number of data flows per layer may help improve data transfer throughput. Fig. 5 shows the throughput of data transfer with compression when a GridFTP *popen* driver [14] is used in the Globus XIO framework. The throughput hits the ceiling of disk I/O bandwidth when the number of threads is 5 since the disk I/O bottleneck indicated by the thin solid line in Fig. 5 gets worse as the number of data reads/threads grows. If we can assign a single thread for disk reads independent of subsequent compression threads, we can maintain the disk I/O bottleneck as constant as a thick solid line in Fig. 5 and we may be able to achieve ideal throughput indicated by the increasing dashed line in Fig. 5.

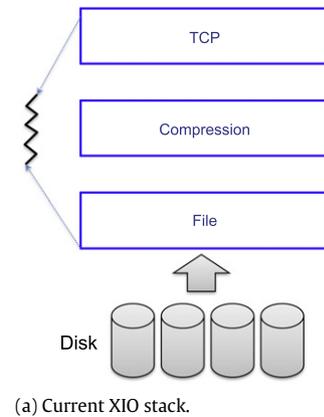
#### 4.2. A novel software architecture for enhanced data transfer throughput

We first present our new architecture and then describe the concrete data flow problem along with optimization algorithms.

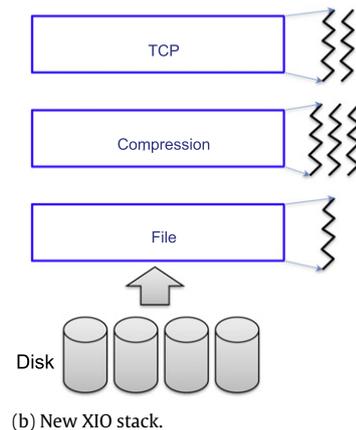
##### 4.2.1. Variable data flows per driver

Basically, the Globus XIO framework is a flexible software I/O stack in which any software drivers corresponding basic I/O functions can be added or removed for a customized use. For example, Fig. 6a shows that the Globus XIO framework is set up with three software drivers such that data are read from file systems and are sent through TCP protocol after compression at the sender side. Even though the Globus XIO framework is mainly used by GridFTP, it can be combined with any applications, which want to send or receive data in a flexible way, as libraries.

The Globus XIO framework is implemented through a registration of four basic I/O functions (*open/close/read/write*) and callback functions. For example, if an application with the Globus XIO stack



(a) Current XIO stack.



(b) New XIO stack.

Fig. 6. Current vs. new XIO stack architecture.

as in Fig. 6a calls an *open()* function, the framework ensures that registered *open()* functions of drivers are called in the order of TCP, Compression, and File layers. It also ensures that callback functions are called in the reverse order if they are set in the course of the previous function calls after *open()* is successfully executed at the bottom driver, i.e., File driver.

However, the current Globus XIO framework does not exploit data parallelism effectively. Current implementation is based on a single thread, which has a lot of room for improvement. Depending on the properties of a driver and system configuration, the optimal number of threads may vary. The optimal number of threads for File driver may be one if a file system is established on a single physical disk since multiple reads from a single disk usually degrade the overall system throughput. In contrast, the optimal number of threads for Compression driver may be equivalent to the number of cores in a system since compression is a compute-intensive job and partitioned data for multiple threads will help improve the overall system throughput.

We propose a novel Globus XIO framework where variable multiple threads are deployed according to properties of drivers as in Fig. 6b. This can be implemented through multi-threaded *read/write* functions of a driver where the number of threads can be adjusted dynamically and each thread will handle its own partitioned data among the whole data. Consequently, the number of threads equals the number of parallel data flows.

#### 4.3. Optimizing data transfer with data compression

In this section, we investigate how variable data flows per layer in the Globus XIO framework affect the overall performance and

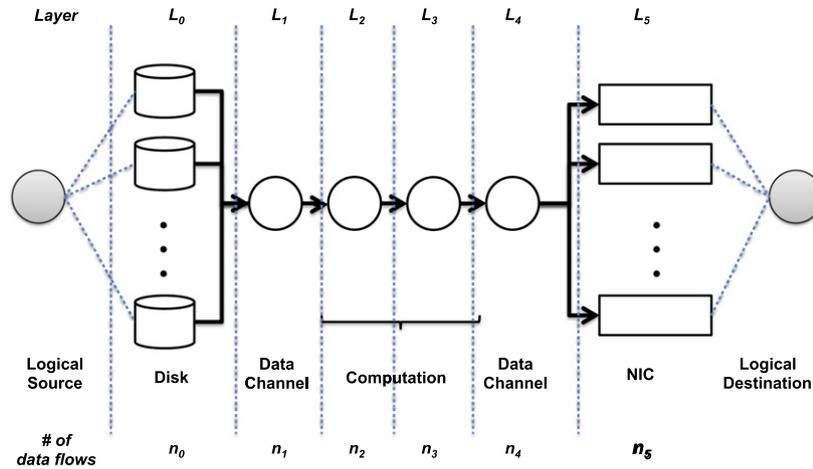


Fig. 7. Variable data flow graph model.

**Table 2**  
Additional notations.

Notation	Description
$N_{src}$	Number of hosts at the source cluster
$N_{dst}$	Number of hosts at the destination cluster
$L_{total}$	Total number of layer
$L_i$	A set of nodes at layer $i$ , $0 \leq i < L_{total}$
$n_i$	Number of data flows at layer $i$ , $0 \leq i < L_{total}$ ; integer variable
$M_i$	Maximum number of data flows at layer $i$ , $0 \leq i < L_{total}$ ; integer variable

how we can optimize the new Globus XIO framework via mathematical formulations. Especially, we focus on a data compression Globus XIO driver where multiple data flows using multi-threads have a large impact on the performance, and develop optimization algorithms for cluster-to-cluster data transfers.

#### 4.3.1. Problem formulation for enhanced data compression

We can extend the formulation for SNDT problem in Fig. 4 such that the extended formulation can take into account variable flows per layer. Fig. 7 shows how we can assign nodes to a layer  $L_i$  and we associate the number of data flows  $n_i$  with the layer  $L_i$ . In Table 2, we list up additional notations including  $n_i$  and  $L_i$  for the formulation.

The complete MICP formulation is described in Fig. 8. Eq. (20) is added compared with the previous formulation in Fig. 4. The number of data flows at each layer  $i$  is defined as  $n_i$ , and bandwidth function is defined by  $n_i$  as in Eq. (19).

## 5. Cluster-wise end-to-end data transfer

In this section, we present an intelligent resource-provisioning mechanism for data transfers from a cluster of size  $m$  to another cluster of size  $n$ , called  $m$ -to- $n$  data transfer.

### 5.1. Problem statement

The cluster-to-cluster data transfer (CCDT) problem is different from the single-node data transfer (SNDT) problem in Section 3. The CCDT problem in this paper takes into account variable data flows per layer and it also seeks the optimal number of hosts at both ends of the clusters, when the maximum numbers of hosts of both clusters are given, as well as the optimal number of variable data flows per layer to achieve maximum data transfer throughput. We assume that all the hosts of a cluster are homogeneous in terms of hardware configuration.

### 5.2. Algorithms for the CCDT problem

Through the algorithm in Fig. 8, we can determine the maximum throughput of a single host at both ends,  $Thr_{src\_single}$ ,  $Thr_{dst\_single}$ , which are the throughput of a source host and the throughput of a destination host, respectively.  $ThrN_{src\_single}$  and  $ThrN_{dst\_single}$  are network throughput corresponding to  $Thr_{src\_single}$  and  $Thr_{dst\_single}$ . Network throughput is the data transfer rate over the network. Thus host throughput is usually greater than network throughput when data compression is deployed. Since we assume homogeneous hosts in a cluster, we can simply match source and destination nodes based on throughput as in Algorithm 1. The network bandwidth,  $BW$ , is also given as an input to the algorithm.

#### Algorithm 1 Node determination algorithm for cluster-to-cluster data transfer (CCDT) problem

**Input:**  $N_{src}$ ,  $N_{dst}$ ,  $Thr_{src\_single}$ ,  $Thr_{dst\_single}$ ,  $ThrN_{src\_single}$ ,  $BW$

**Output:**  $n_{src}$ ,  $n_{dst}$

```

1:  $n_{src}$ ,  $n_{dst} \leftarrow 0$ 
2: while  $n_{dst} < N_{dst}$  do
3:    $n_{dst} \leftarrow n_{dst} + 1$ 
4:   if  $Thr_{dst\_single} < Thr_{src\_single}$  then
5:     if  $BW - ThrN_{src\_single} < 0$  then
6:       return;
7:     end if
8:      $n_{src} \leftarrow n_{src} + 1$ 
9:      $BW \leftarrow BW - ThrN_{src\_single}$ 
10:  else
11:    Increase  $n_{src}$  up to by  $\lfloor Thr_{dst\_single} / Thr_{src\_single} \rfloor$  while
    decreasing  $BW$  in the same way in lines 5 through 9.
12:  end if
13: end while

```

## 6. Experimental evaluation

### 6.1. Disk to disk transfer on high-speed networks

We have conducted experiments on an ESnet 100G testbed [24] in two locations: NERSC (Oakland, CA) and StarLight (Chicago, IL). Fig. 9 shows the detailed configuration of the testbed. At NERSC, there are 5 hosts of three different hardware configurations. Three hosts, nersc-diskpt-1, nersc-diskpt-2, and nersc-diskpt-3, have Intel Xeon Nehalem E5650 ( $2 \times 6 = 12$  cores), multiple 10G NICs, and 4 RAID 0 sets of 4 drives. The other two hosts do not have RAID drives but have only a local disk. On the other hand, there are 3

**Objective**

$$\text{maximize } T \quad (16)$$

**Subject to:**

$$r_{lk} \geq 0, (l, k) \in E \quad (17)$$

$$0 \leq n_i \leq M_i, 0 \leq i < L_{total} \quad (18)$$

$$r_{lk} \leq \begin{cases} B_{lk}^d(n_i), (l, k) \in E, i \in L_i & \text{if } V_l \text{ is a disk node} \\ B_{lk}^c(n_i), (l, k) \in E, i \in L_i & \text{if } V_l \text{ is a computation node} \\ B_{lk}^n(n_i), (l, k) \in E, i \in L_i & \text{if } V_l \text{ is a NIC node} \end{cases} \quad (19)$$

$$\sum_{k:(l,k) \in E} r_{lk} - c \sum_{k:(k,l) \in E} r_{kl} = 0, \quad (20)$$

$$l \neq s_j, l \neq d_j, c = \begin{cases} \text{compression ratio,} & \text{if } l \text{ is compression node} \\ 1, & \text{otherwise} \end{cases}$$

$$\sum_{k:(s,k) \in E} r_{sk} - \sum_{k:(k,s) \in E} r_{ks} \geq T \quad (21)$$

$$\sum_{k:(k,d) \in E} r_{ks} - \sum_{k:(d,k) \in E} r_{dk} \geq T \quad (22)$$

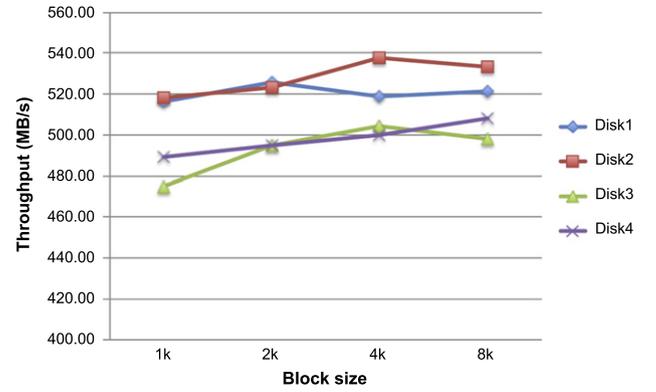
$$\sum_{k:(l,k) \in E} C(r_{lk}) \leq \min(N_c \times 100, n_s \times N_d \times 100) \quad (23)$$

**Fig. 8.** Variable flow optimization algorithm.

hosts without disk arrays at StarLight. These hosts have 2 AMD 6140 ( $2 \times 8 = 16$  cores) and multiple 10G NICs, but do not have RAID disks. The hosts at StarLight have only local disks, which are slow (i.e.,  $\sim 300$  MB/s) and thus cannot saturate even a 10G link. For this reason, we conducted disk-to-memory tests where all data flows departing from hosts at NERSC are directed to `/dev/null` on hosts at StarLight so that we can assume that the hosts at StarLight have the same disks as those of the hosts at NERSC.

We have chosen various sizes of datasets including lots of small files (LOSF) dataset for evaluation of optimizing the end-to-end data transfer rates. We use four different datasets—ten thousands of 1 MB files, one thousand 10 MB files, one hundred 100 MB files, and ten 1 GB files such that total amount of each dataset would be around 10 GB. The files were synthetically generated using `/dev/urandom` in Linux.

To measure the disk performance, we use `dd` and `iozone` [25] as disk I/O benchmark tools. In addition, we use `nmon` [26] and

**Fig. 10.** Disk throughput at NERSC using `dd`: similar disk throughput with varying block size.

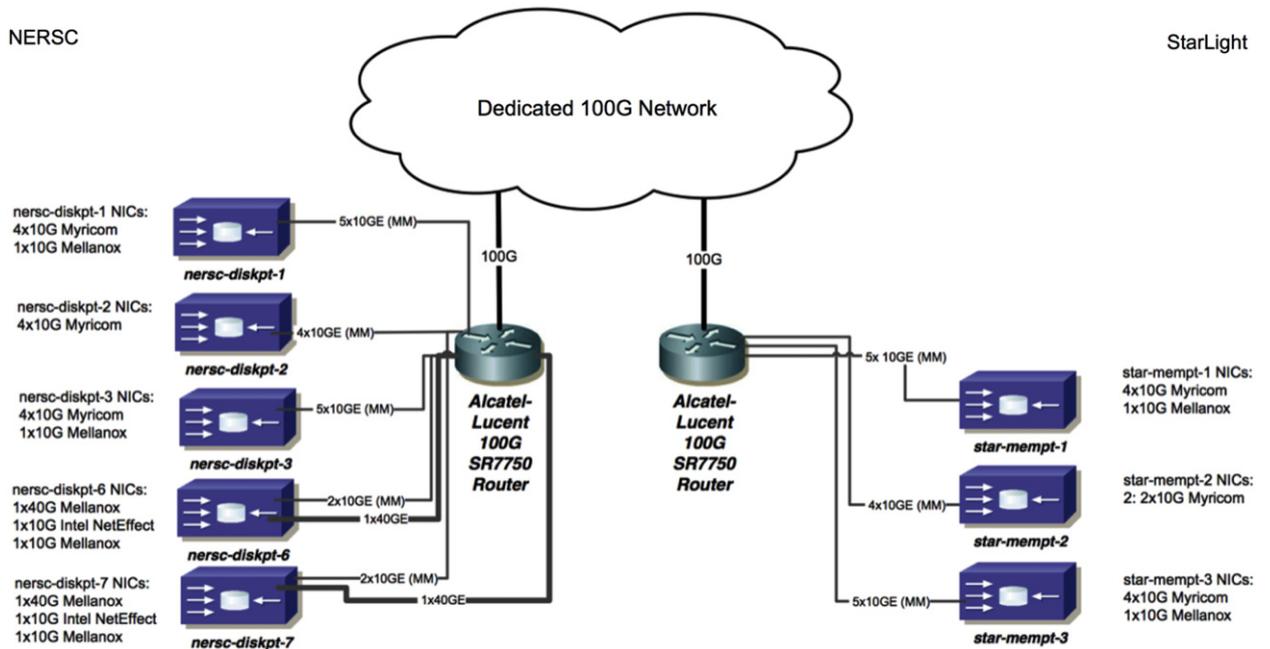
`netperf` [27] as benchmark tools to measure CPU load and network performance, respectively.

**6.1.1. Subsystem tests for model parameter setting**

We first conducted basic disk I/O performance tests using `dd` disk utility to obtain baseline performance of disk throughputs. Fig. 10 shows the disk read throughputs ( $\sim 500$  MB/s) of 4 RAID sets attached to hosts at NERSC. The theoretical upper limit of each RAID disk is around 1.2 GB/s since the RAID disk is composed of four disks with 300 MB/s read performance. Even though there are performance variances among disks, we ignore the variances for simplicity in this paper.

Next, we measured the multithread disk read performance depending on file size and the number of threads to determine the value of  $\alpha, \gamma$  in Eq. (3). Fig. 11 shows that disk throughput decreases as the number of streams increases regardless of applications' read unit sizes (i.e., 1 MB and 10 MB). We conducted experiments in case of sequential disk read. Eq. (3) determined by these results would be  $B_{lk}^d(\cdot)$  in Fig. 4 where  $n_s$  equals  $L$ , and  $l$  is a disk node.

However, as Fig. 12 shows, the aggregate disk throughput using multiple disks does not scale linearly due to channel contention.

**Fig. 9.** ESnet 100G testbed.

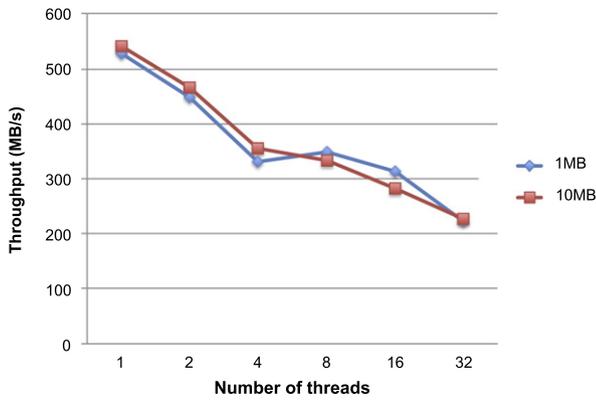


Fig. 11. Disk throughput at NERSC using *iozone*: decreasing disk throughput with increasing number of data streams.

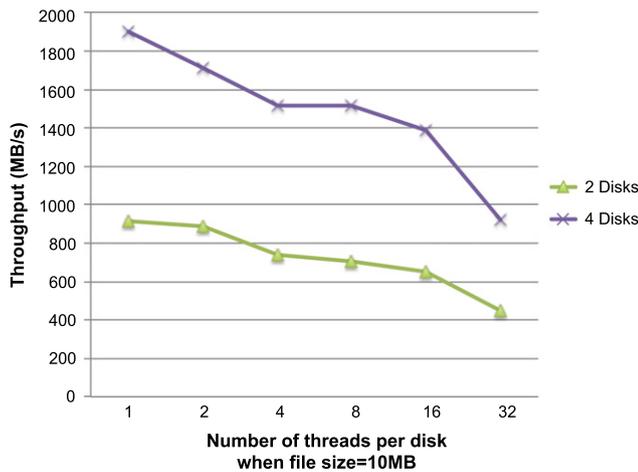


Fig. 12. Multiple disk throughput at NERSC using *iozone*: increased throughput using multiple disks, but slightly under the total sum of individual disks.

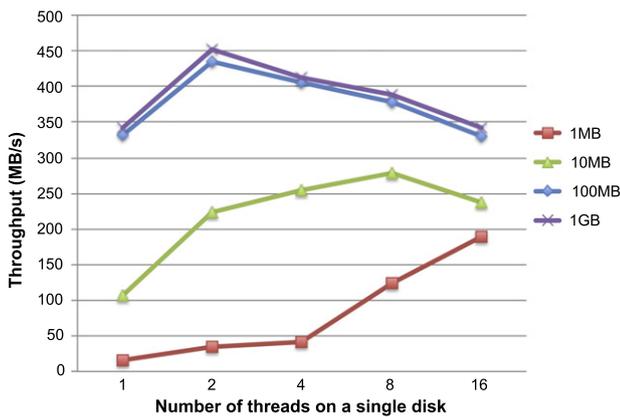


Fig. 13. GridFTP throughput: increasing throughput until disk throughput or data contention among multiple data streams becomes a bottleneck.

We model the channel contention using a data channel node and associated bandwidth function as in Eq. (3).

We measured the application (i.e., GridFTP) throughput while varying the number of data streams as in Fig. 13. We can model GridFTP throughput through Eq. (4) by ignoring the decreasing throughput after hitting the peak because that is due to disk bottleneck which is already modeled by *disk edges*. Even though the data movement tool GridFTP is the only application used for the end-to-end data transfers in this paper, we can model any applications such as compression in a similar way through Eqs. (4) and (5).

Regarding network edges, Fig. 14a and b shows the throughput and CPU load of TCP protocol, respectively. We conducted memory-to-memory transfer using a single 10 Gb NIC, and we measured these TCP performance results by *netperf*. Fig. 14a shows that network transfer throughput is saturated with 3 TCP streams, and is near the full capacity of the 10 Gb NIC. Without 10 Gb capacity limitation, the TCP throughput with 3 TCP streams should be beyond 10 Gb. Fig. 14b also shows that CPU load is extraordinarily high near 10 Gb/s due to contention among 3 streams. We note that 100% CPU corresponds to a full usage of one CPU out of multiple CPUs in a host. Even though such non-linear behaviors happen, we can capture them using linear regression (i.e., Eqs. (6) and (7)) since this is simple and more conservative. For example, the linear function that captures TCP CPU load in Fig. 14 will overestimate the CPU load for TCP traffic more than the actual CPU load.

### 6.1.2. Results and discussion

We have compared our model-based optimization approach with two cases: (1) GridFTP with only-*fast option*, (2) GridFTP with auto-tuning optimizations currently used by Globus Online [9], as these are the commonly used approaches by the end users for GridFTP data movement. Globus Online's auto-tuning algorithm uses different GridFTP optimization options depending on file size. If the number of files is more than 100 and an average file size smaller than 50 MB, it uses GridFTP with concurrency = 2 files, parallelism = 2 sockets per file, and pipelining = 20 requests outstanding at once. If the file size is larger than 250 MB, Globus Online uses options of concurrency = 2, parallelism = 8, and pipelining = 5. In all other cases, the default setting is used: concurrency = 2, parallelism = 4, and pipelining = 10.

Fig. 15 shows the experimental results of all three cases. The data transfer experiments have been done from NERSC to StarLight by varying the number of hosts at NERSC from 1 to 5 and the number of hosts at StarLight from 1 to 3. The numbers of hosts at NERSC and StarLight are kept same except when the number of hosts at NERSC is greater than 3. In such cases, the number of hosts at StarLight is set to 3. The disk-to-disk data transfer on this testbed is bottlenecked by disks while each host has multiple 10 Gb/s NICs and the wide-area network links have enough bandwidth of 100 Gb/s. In such cases, we can achieve higher throughput by utilizing multiple hosts together with optimized data transfer. We compute the data transfer throughput by measuring the total time taken for transferring certain datasets. Globus Online outperforms the naive GridFTP especially in the cases of 1 MB and 10 MB datasets. Our model-based optimizations are 3–4 times faster than Globus Online in most cases, and 8 times faster than Globus Online, particularly, in the case of 1 MB datasets. It is mainly because our model can effectively identify the number of data flows based on disk throughput performance models and utilize data flow parallelism through multiple disks. For instance, with  $-cc = 2$  options, Globus Online can utilize only two data streams from disks, which has a lot room for improvement, and cannot utilize the advantages of multiple disks. Based on models, our formulation in Fig. 4 could find the proper number of data flows, 8, 6, 3, 2 in the case of 1 MB, 10 MB, 100 MB, 1 GB files, respectively. The data transfer throughput scales well as the number of hosts at NERSC increases up to 3 hosts. In case of 4 and 5 hosts at NERSC, the increasing rate slows down because those hosts have only local disks. In addition, our formulation could find a solution suggesting using multiple NICs in case that the aggregate throughput is beyond the capability of a 10G NIC.

The disk-to-disk data transfer on the ESnet 100G testbed suffers from disk bottlenecks in case of big files over 100 MB as shown in Fig. 13. The emerging high-performance solid state storages (SSDs) can improve the overall data transfer throughput. The high-performance SSDs can affect our model-based optimization

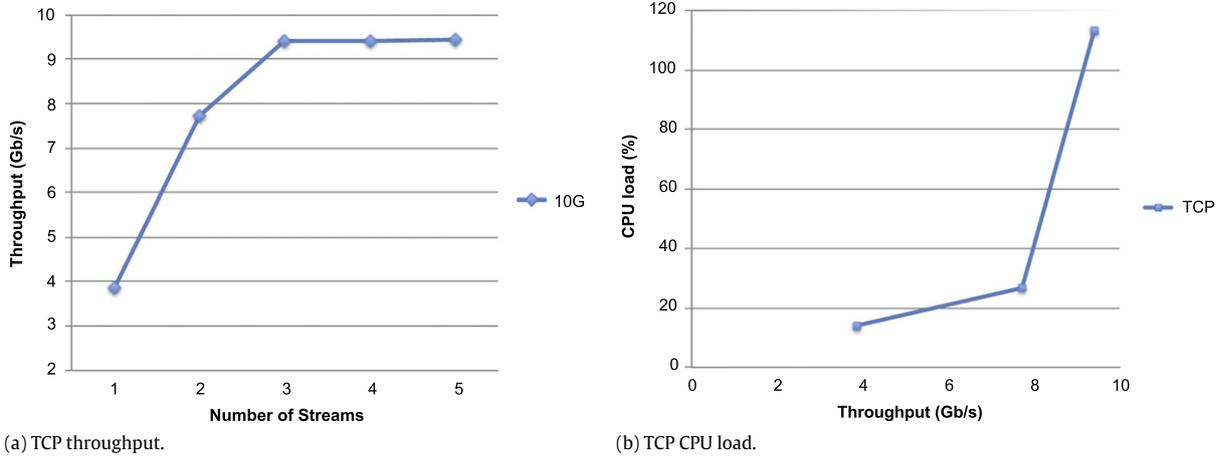


Fig. 14. TCP protocol characteristics.

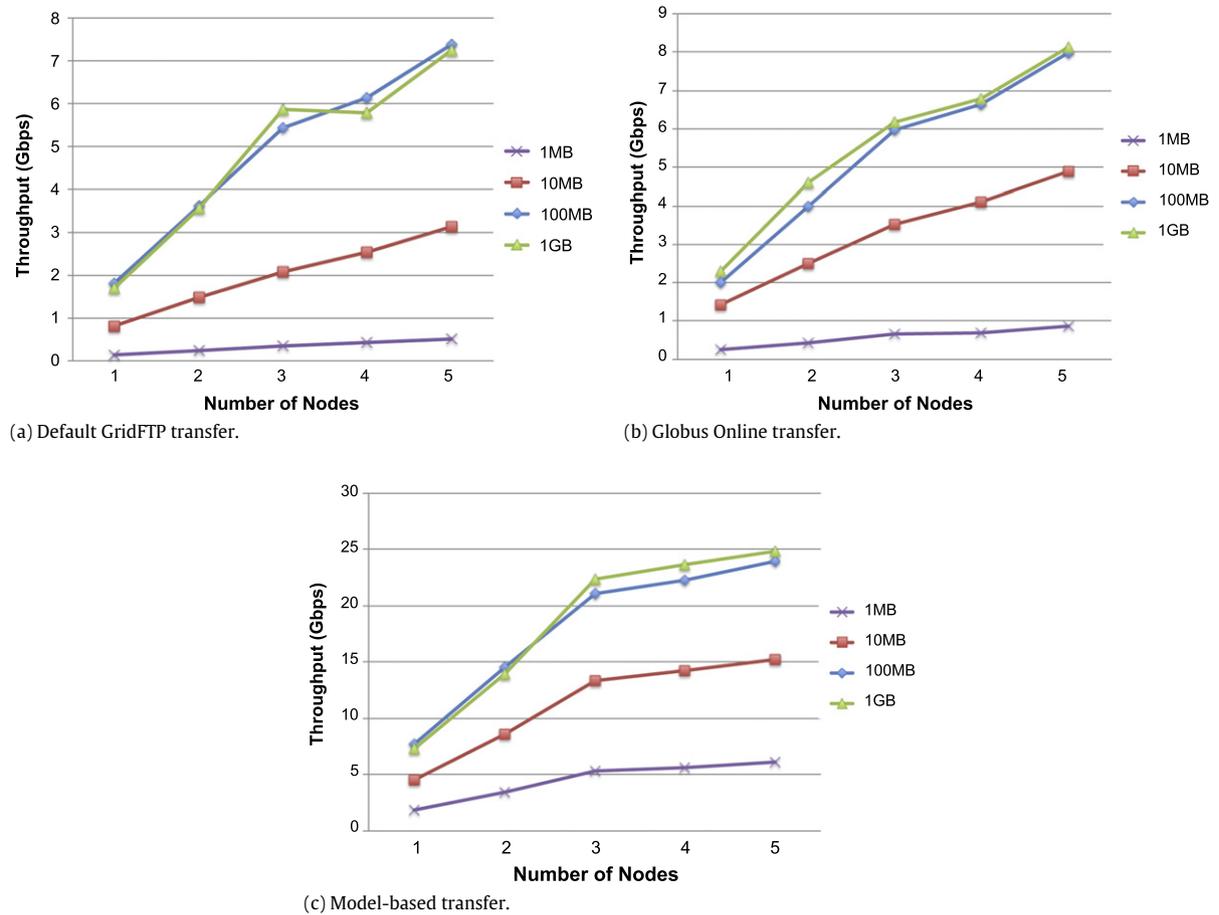


Fig. 15. Data transfer throughput comparison.

approach in two ways. First, the disk performance model for SSD is different from the model for HDD. In case of HDD RAID disks in our paper, the read performance decreases as the number of threads increases as shown in Figs. 11 and 12. The read performance of SSDs would show less performance degradation with more numbers of threads since an SSD does not have a mechanical arm to read and write data and accordingly needs less wait time between consecutive disk service requests. The read performance of SSDs can still be modeled by the same equation as Eq. (1). In contrast, it is more complicated to model the write performance of SSDs than HDD because a write operation of SSDs is involved in additional erase and wear-level management operations [28].

However, we can also model the write performance as long as there are analytical or empirical models that can be approximated by linear functions. Second, the high-performance SSDs will make the disk part not bottlenecked any more in overall disk-to-disk data transfer. Consequently, the high-performance SSDs will result in less numbers of disks and more other resources such as CPU and network to use to achieve higher data transfer throughput than HDDs.

The advantages of using model-based optimization formulations are as follows: (1) it can suggest the future hardware plan optimized for overall data transfer throughput just by simulating different configurations of hardware as well as software, (2) it can

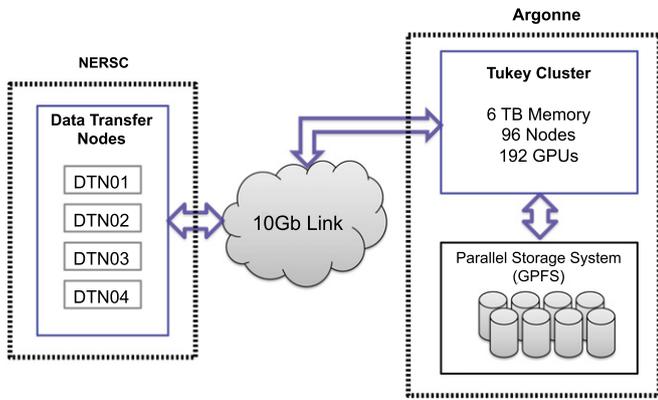


Fig. 16. Data transfer infrastructure.

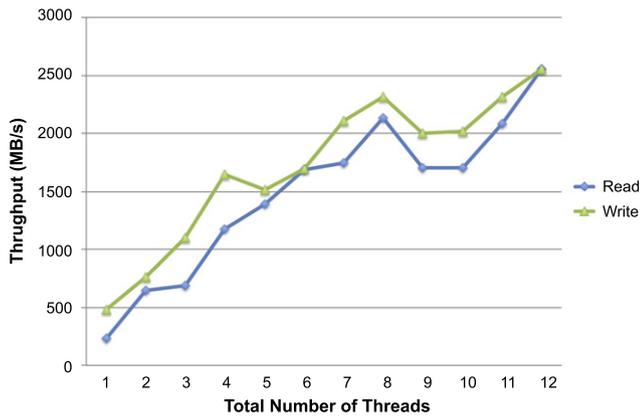


Fig. 17. Storage system benchmark via IOR at NERSC.

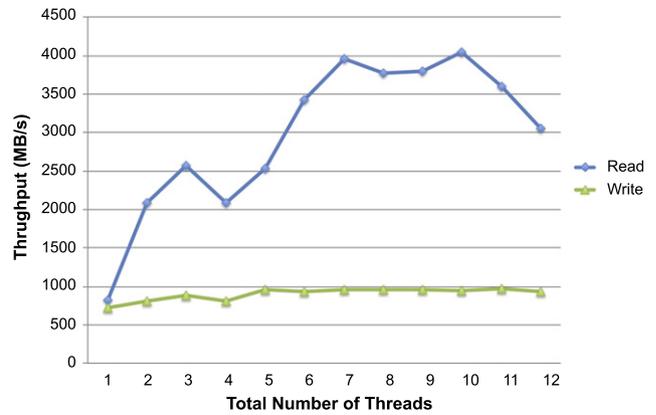


Fig. 18. Storage system benchmark via IOR at Tukey.

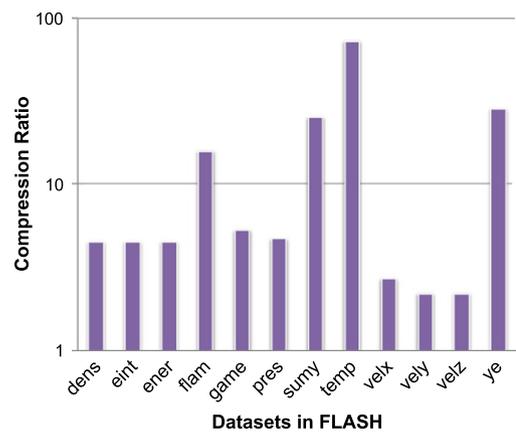


Fig. 19. *blosc* compression ratio with regard to FLASH datasets [19].

be used by systems such as Globus Online and other intelligent data transfer managers to adaptively optimize transfers for varying CPU resource availability and network status, and (3) it can provide basic models for simulating bulk data movement in the next generation networks.

## 6.2. Cluster-wise data transfer with data compression

We have performed experiments on data transfer nodes (DTNs) at NERSC [29] and Tukey at Argonne as in Fig. 16. There are four DTNs at NERSC and Tukey has a total of 96 compute nodes, where each node has 16 CPU cores and two NVIDIA Tesla M2070 GPUs. Data are transferred from Tukey to NERSC where the number of hosts at Tukey varies from 1 to 4 and the number of hosts at NERSC varies from 1 to 12. One thing that is distinguished from the previous testbed in Fig. 9 is that the storage systems of both clusters are shared parallel file systems.

We use FLASH, an astrophysics simulation, datasets to evaluate cluster-wise data transfer with data compression. FLASH datasets consist of several datasets, and we pick three datasets such as *temp*, *pres*, and *velx*, which represent high compression ratio dataset, middle compression ratio dataset, and low compression ratio dataset, respectively.

### 6.2.1. System modeling for parallel file systems and compression

The testbed in Fig. 16 is different from the previous testbed in Fig. 9 in two aspects. First, the new testbed has parallel file systems as storage systems of both clusters. Second, data compression computation is deployed for improving overall data transfer throughput in case of network bottleneck.

We first measured performance of shared file systems at both clusters using IOR [30], a parallel file system benchmark tool.

Figs. 17 and 18 show the file system performance, when file read/write size is 10 MB, at NERSC and Tukey, respectively. Only read performance is considered at Tukey whereas only write performance is considered at Tukey since data are transferred from Tukey to NERSC. The storage system performance is modeled using linear or quadratic regression since there is no well-known mathematical analytic model for parallel file systems.

We use the *blosc* compression algorithm [31] which has better performance compared with typical compression algorithms in terms of both compression ratio and compression time. Fig. 19 shows that the *blosc* compression ratios of FLASH/*temp*, FLASH/*pres*, and FLASH/*velx* are 69.89, 4.49, and 2.46, respectively. We also measured *blosc* compression/decompression throughput by varying the number of threads up to the number of cores at the host to model the throughput using linear functions. We noticed that the throughput decreases beyond a certain point due to internal *blosc* compression algorithm operations as in Fig. 20, and we set the maximum number of threads to that point (e.g. 8 at Tukey) in order to model the throughput simply using linear functions.

### 6.2.2. Results and discussion

We first measured the performance of data transfer between single hosts at each cluster in order to compare with the performance of cluster-to-cluster data transfer. We then measured the performance of cluster-to-cluster data transfer by increasing the number of data transfer processes from 1 to 12 at each cluster regarding three datasets, i.e., FLASH/*temp*, FLASH/*pres*, and FLASH/*velx*. Only one data transfer process is assigned to one host at Tukey since one process is enough to achieve the maximum throughput of one host, which will be explained in the following.

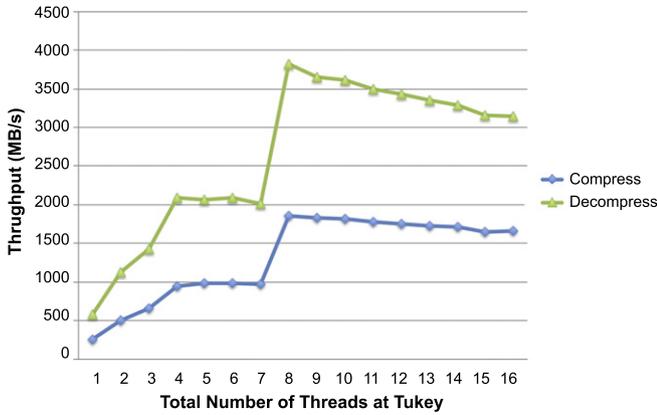


Fig. 20. *blosc* throughput in terms of the number of threads regarding random data.

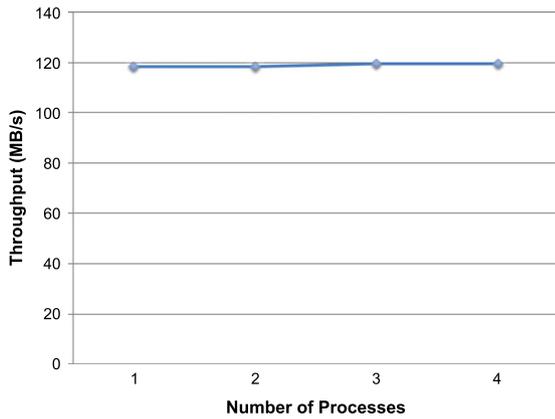


Fig. 21. Memory-to-memory data transfer between single hosts.

On the other hand, multiple data transfer processes are assigned to one host at NERSC in a round-robin fashion when the number of process is beyond 4 since the number of DTNs at NERSC is limited to 4. We also varied the number of compression threads from 1 to 8 to show how the parallelism of data flow affect the overall data transfer throughput. Lastly, we discuss our variable optimization algorithms' solutions with respect to the measured performance results.

Fig. 21 shows the maximum data transfer throughput between single nodes at each cluster. This is a memory-to-memory data transfer result without data compression and the throughput is limited by capacity of NICs, 1 Gb/s (~120 MB/s) of hosts at Tukey.

The experimental results of memory-to-memory data transfer when multiple nodes are utilized are shown in Fig. 22 where *Normal* indicates data transfer without data compression and *CompressionX* indicates data transfer with data compression using *X* threads. The results show that the performance scales well with regard to the number of nodes at Tukey in all cases. Since the network link between NERSC and Tukey is 10 Gb/s (~1.2 GB/s), *normal* data transfer seems to achieve the maximum throughput using 12 hosts at Tukey. The results also show that the performance of all three FLASH datasets is similar when one thread is used for compression while the throughput of the FLASH/velx dataset with 8 threads is almost 1.5 times better than the throughput with 1 thread. Compared with the normal case, the throughput of the FLASH/velx dataset is 8 times better, which means data transfer with optimized data compression helps improve throughput drastically in case of limited network bandwidth and/or NIC capacities.

Figs. 23 through 25 show disk-to-disk data transfer throughputs. All the results show that the throughput of normal disk-to-disk data transfer does not scale after 10 nodes and even decreases.

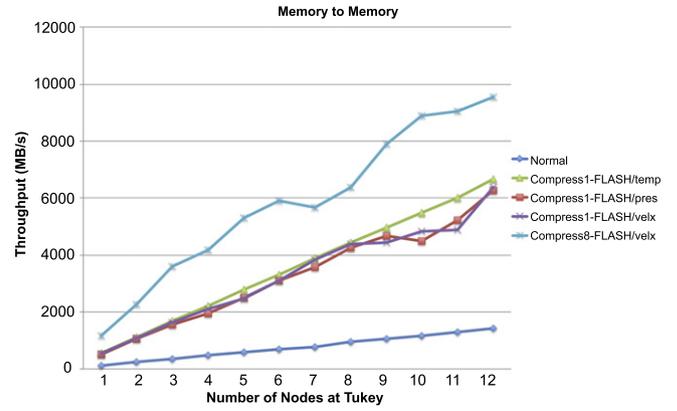


Fig. 22. Memory-to-memory data transfer.

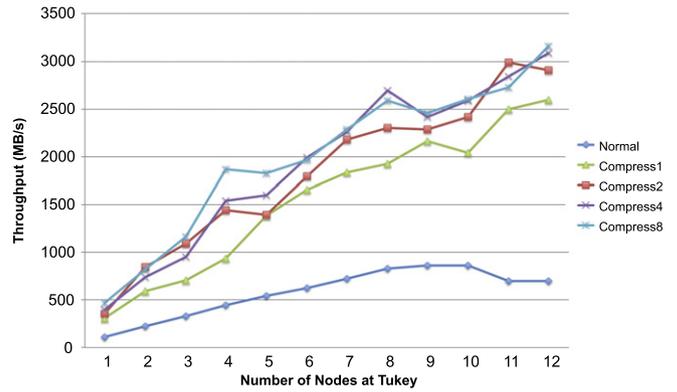


Fig. 23. Disk-to-disk data transfer (FLASH/temp).

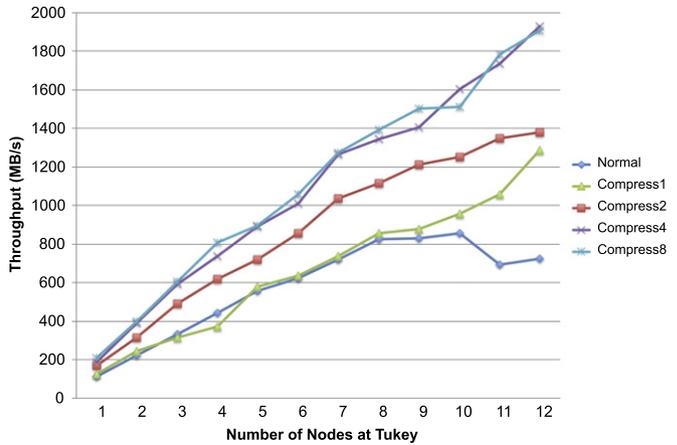


Fig. 24. Disk-to-disk data transfer (FLASH/pres).

Considering that the 10 Gb/s link between NERSC and Tukey is shared by many users and the memory-to-memory transfer experiments were conducted at different time periods, the performance decrease in case of normal disk-to-disk data transfer is mainly due to network link bottleneck. Such situations can be seen in the case of FLASH/velx data transfer with 6 or 8 compression threads. Since the compression ratio of FLASH/velx is 2.46, the data transfer of FLASH/velx with data compression saturates the network link when the number of nodes is 10. In case of other datasets such as FLASH/temp and FLASH/pres, data transfers do not saturate the network link due to higher compression ratio and scale well with regard to the number of nodes.

We compare the experimental results with optimized throughput based on our algorithms in Fig. 8 and Algorithm 1. Fig. 26

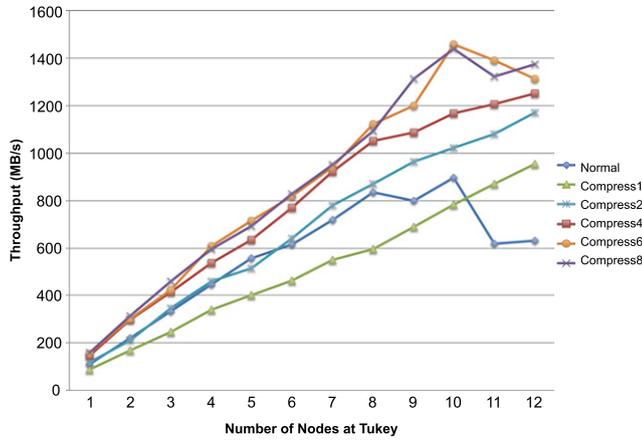


Fig. 25. Disk-to-disk data transfer (FLASH/velx).

shows the data flow graph model for the CCDT problem. Using formulations in Fig. 8, we can get the numbers of compression threads, 8, 4, and 6, for FLASH/temp, FLASH/pres, and FLASH/velx, respectively. In addition, we can get the maximum throughput for single-host data transfer. These numbers are used to compute the required number of hosts on each cluster. We can get the numbers of hosts at NERSC and Tukey, (4, 12), (4, 12) and (3, 10) for FLASH/temp, FLASH/pres, and FLASH/velx, respectively. In particular, (3, 10) for FLASH/velx can achieve 1.3 GB/s, which is comparable to throughput of (4, 10) in the experiments. The solutions based on our approach are consistent with experimental results. Please note that DTNs at NERSC are in production and shared by many data transfers; we used the network bandwidth between NERSC and Argonne probed just before the algorithms are run to make sure the results reflect the current system status.

6.3. Deployment in real systems

As described in the paper, we need two steps to get the optimized solution. First, we have to build system models by running some system benchmarks such as disk throughput and network throughput measurement. Even though they take long time (a few hours up to one or two days), the benchmark tests need to be performed only when there is a system change such as new disk installation. In addition, this step can be automated through scripts if needed. The dynamic factors such as compression rates need to

be updated more often, but even in such cases, the ratio of the time spent on tests to the time taken for transfer will be quite small and the tests can be done concurrently when other data transfers are being fulfilled. If the number of parameters affecting the performance is high, and exhaustive benchmark testing to explore the whole search space is not possible, we can use surrogate models such as support vector machines to predict with small sets of collected data, which is out of scope of the paper. Second, we have to run optimization solver such as CPLEX [22] to get the solution. Since the problem size is small (the graph consists of tens of nodes and tens of edges), the running time is a few seconds, which is within reasonable time window. Overall, the proposed method is useful in practice.

Regarding interplay of flows, we can think of the following three cases. As for interplay of flows in disk I/O, processes are usually allocated fair shares of CPU, and hence they have equal probability of issuing disk I/O requests. However, if processes other than disk I/O processes in our model occupy the most resources (i.e. overloaded state) or share resources with processes in our model, disk I/O processes would not behave as models suggest. The more sophisticated models for disk I/O considering system-wide circumstances are needed for accurate modeling. Interplay in computing can also be addressed by similar approaches. Regarding interplay in networks, TCP streams are believed to fairly share a network link. However, since other network traffics are injected into the shared networks, similar circumstances to disk I/O can happen. In order to cope with such situations, the network status needs to be probed regularly to reflect dynamic variance of available network bandwidths.

Regarding data compression, data may be composed of multiple sets of sub-data with different characteristics, it is true that the current model and formulation cannot cope with such situations. To address such issues, the data to transfer should be evaluated on the fly to estimate the data compression ratios, and the formulation should introduce separate compression ratio parameters for each flows, which means separate computation nodes for flows from different disks.

7. Conclusions

We first model all the system components involved in end-to-end data transfer as a graph. We then formulate the problem whose goal is to achieve maximum data transfer throughput using parallel data flows. We also propose a new I/O stack with variable

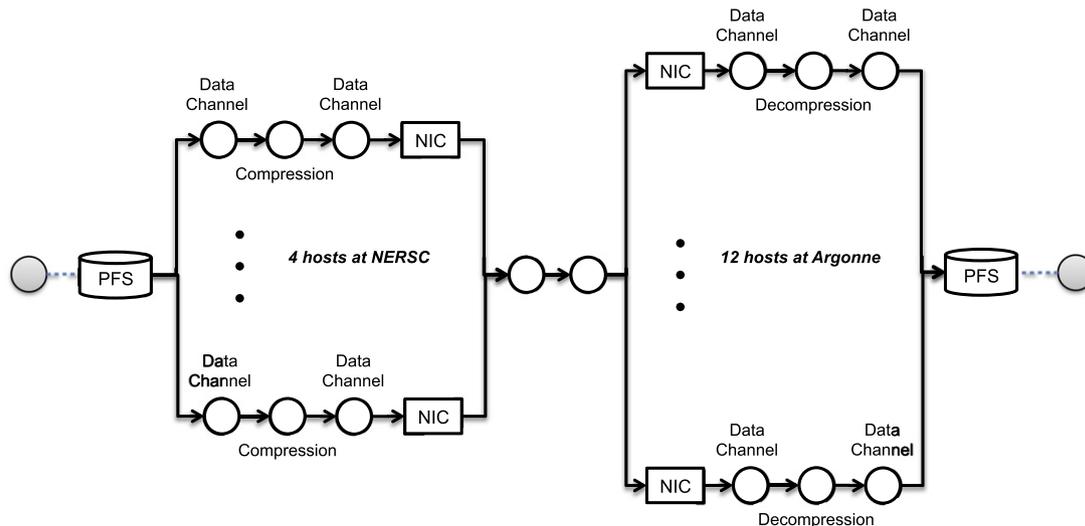


Fig. 26. Data flow graph model for the CCDT problem.

data flows and cluster-wise optimization algorithms to enhance data transfer throughput using data compression. Our proposed formulations and solutions are evaluated through experiments on the two different testbeds, the ESnet 100G testbed and a cluster data transfer testbed between NERSC and Argonne. The experimental results on the ESnet 100G testbed show that our approach is around four times faster than Globus Online for most datasets. The experimental results on the cluster data transfer testbed show that the throughput of cluster data transfer with data compression is up to four times faster than the throughput of normal cluster data transfer and 10 times to 20 times faster than single host data transfer. It is promising that flexible throughput optimization algorithms can detect the performance bottleneck and can suggest parameters such as the number of nodes and whether or not to use compression. This work can be automated through periodic system profiling and resource provisioning based on the proposed optimization algorithms.

## Acknowledgment

This work was supported by the US Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

## References

- [1] A. Rajendran, P. Mhashilkar, H. Kim, D. Dykstra, I. Raicu, Optimizing large data transfers over 100 Gbps wide area networks, 2012.
- [2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The globus striped GridFTP framework and server, in: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC'05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 54–64. <http://dx.doi.org/10.1109/SC.2005.72>.
- [3] A. Shoshani, A. Sim, J. Gu, Grid Resource Management, Kluwer Academic Publishers, Norwell, MA, USA, 2004, pp. 321–340.
- [4] W. Allcock, J. Bresnahan, K. Kettimuthu, J. Link, The Globus extensible input/output system (XIO): a protocol independent IO system for the grid, in: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, 2005, pp. 8–15. <http://dx.doi.org/10.1109/IPDPS.2005.429>.
- [5] E. Kissel, M. Swany, Evaluating high performance data transfer with RDMA-based protocols in wide-area networks, in: Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication, HPCC'12, Washington, DC, USA, 2012, pp. 802–811. <http://dx.doi.org/10.1109/HPCC.2012.113>.
- [6] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, I. Foster, Gridftp pipelining, 2007.
- [7] R. K., et al., Lessons learned from moving Earth system grid data sets over a 20 Gbps wide-area network, in: 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10, ACM, New York, NY, USA, 2010, pp. 316–319. <http://dx.doi.org/10.1145/1851476.1851519>.
- [8] Globus toolkit website, <http://globus.org/toolkit/>, 2013. Last visited on Nov.11.2013.
- [9] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, S. Tuecke, Software as a service for data scientists, *Commun. ACM* 55 (2012) 81–88.
- [10] E. Yildirim, J. Kim, T. Kosar, How GridFTP pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers, in: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:, 2012, pp. 506–515. <http://dx.doi.org/10.1109/SC.Companion.2012.73>.
- [11] D. Gunter, R. Kettimuthu, E. Kissel, M. Swany, J. Yi, J. Zurawski, Exploiting network parallelism for improving data transfer performance, in: High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:, 2012, pp. 1600–1606. <http://dx.doi.org/10.1109/SC.Companion.2012.337>.
- [12] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, I. Foster, GridFTP pipelining, Teragrid, 2007.
- [13] J. Bresnahan, I. Foster, An architecture for dynamic allocation of compute cluster bandwidth (MS thesis), Department of Computer Science, University of Chicago, 2006.
- [14] R. Kettimuthu, S. Link, J. Bresnahan, M. Link, I. Foster, Globus XIO pipe open driver: enabling GridFTP to leverage standard unix tools, in: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, TG'11, ACM, New York, NY, USA, 2011, pp. 20:1–20:7. <http://dx.doi.org/10.1145/2016741.2016763>.
- [15] R. Kettimuthu, W. Liu, J.M. Link, J. Bresnahan, A GridFTP transport driver for Globus XIO, in: International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA, 2008, pp. 843–849.
- [16] Eun-Sung Jung, Rajkumar Kettimuthu, Venkatram Vishwanath, Toward optimizing disk-to-disk transfer on 100G networks, in: IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), ANTS'13, IEEE Computer Society, Chennai, India, 2013.
- [17] E. Yildirim, T. Kosar, End-to-end data-flow parallelism for throughput optimization in high-speed networks, *J. Grid Computing* 10 (2012) 395–418.
- [18] B. Welton, D. Kimpe, J. Cope, C.M. Patrick, K. Iskra, R.B. Ross, Improving I/O forwarding throughput with data compression, in: CLUSTER'11, 2011, pp. 438–445.
- [19] H. Bui, V. Vishwanath, H. Finkel, K. Harms, J. Leigh, S. Habib, K. Heitmann, M.E. Papka, Scalable parallel I/O on Blue Gene/Q supercomputer using compression, topology-aware data aggregation, and subfling, in: The Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, 2014.
- [20] T. Bicer, J. Yin, D. Chiu, G. Agrawal, K. Schuchardt, Integrating online compression to accelerate large-scale data analytics applications, in: 2013 IEEE 27th International Symposium on Parallel Distributed Processing, IPDPS, 2013, pp. 1205–1216.
- [21] E.K. Lee, R.H. Katz, An analytic performance model of disk arrays, *SIGMETRICS Perform. Eval. Rev.* 21 (1993) 98–109.
- [22] cplex website, <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, 2013. Last visited on Nov.11.2013.
- [23] T. Hacker, B. Athey, B. Noble, The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network, in: Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, 2002, pp. 10–19. <http://dx.doi.org/10.1109/IPDPS.2002.1015527>.
- [24] ESnet 100G testbed, <http://www.es.net/RandD/100g-testbed/>, 2013. Last visited on Nov.11.2013.
- [25] iozone website, <http://www.iozone.org/>, 2013. Last visited on Nov.11.2013.
- [26] nmon website, <http://nmon.sourceforge.net/>, 2013. Last visited on Nov.11.2013.
- [27] netperf website, <http://www.netperf.org/netperf/>, 2013. Last visited on Nov.11.2013.
- [28] P. Desnoyers, Analytic modeling of SSD write performance, in: Proceedings of the 5th Annual International Systems and Storage Conference, SYSTOR'12, ACM, New York, NY, USA, 2012, pp. 1–10. <http://dx.doi.org/10.1145/2367589.2367603>.
- [29] NERSC website, <http://www.nersc.gov>, 2013. National Energy Research Scientific Computing Center, Last visited on Nov.11.2013.
- [30] IOR website, <http://sourceforge.net/projects/ior-sio/>, 2013. Last visited on Nov.11.2013.
- [31] blocs website, <http://blocc.pytables.org/trac>, 2013. Last visited on Nov.11.2013.



tems. He has published over 10 papers in conference proceedings and journals.



ACM.

**Eun-Sung Jung** is a postdoctoral researcher in the Mathematics and Computer Science Division at Argonne National Laboratory. He earned a Ph.D. from the Department of Computer and Information Science and Engineering at the University of Florida. He also received B.S. and M.S. degrees in electrical engineering from Seoul National University, Korea, in 1996 and 1998, respectively. He also held a position of a research staff member at Samsung Advanced Institute of Technology from 2011 to 2012. His current research interests include cloud computing, network resource/flow optimization, and real-time embedded systems.

**Rajkumar Kettimuthu** is a project leader in the Mathematics and Computer Science Division at Argonne National Laboratory and a fellow at University of Chicago's Computation Institute. His research interests include transport protocols for high-speed networks; research data management in distributed systems; and the application of distributed computing to problems in science and engineering. He is the technology coordinator for Globus GridFTP, a widely used data movement tool. He has published over 60 articles in parallel, distributed and high performance computing. He is a senior member of IEEE and



ACM.

**Venkatram Vishwanath** is an Assistant Computer Scientist in the Mathematics and Computer Science Division at Argonne National Laboratory. He is also a member of the Argonne Leadership Computing Facility. He joined Argonne as an Argonne Scholar and Argonne Director's Fellow in 2009. His areas of research include runtime and programming models for data-intensive computing, scalable algorithms for data movement, scientific data visualization, and performance analysis for parallel applications. He completed his doctorate degree in computer science in 2009 from the University of Illinois at Chicago. He is also a Fellow of the Computation Institute at the University of Chicago and an adjunct professor in the Department of Computer Science at the Northern Illinois University.