Design and Evaluation of a Simple Data Interface for Efficient Data Transfer Across Diverse Storage

ZHENGCHUN LIU, Argonne National Laboratory
RAJKUMAR KETTIMUTHU, Argonne National Laboratory
JOAQUIN CHUNG, Argonne National Laboratory
RACHANA ANANTHAKRISHNAN, The University of Chicago
MICHAEL LINK, The University of Chicago
IAN FOSTER, Argonne National Laboratory and The University of Chicago

Modern science and engineering computing environments often feature storage systems of different types, from parallel file systems in high-performance computing centers to object stores operated by cloud providers. To enable easy, reliable, secure, and performant data exchange among these different systems, we propose `Connector`, a plug-able data access architecture for diverse, distributed storage. By abstracting low-level storage system details, this abstraction permits a managed data transfer service (Globus in our case) to interact with a large and easily extended set of storage systems. Equally important, it supports third-party transfers: that is, direct data transfers from source to destination that are initiated by a third-party client but do not engage that third party in the data path. The abstraction also enables management of transfers for performance optimization, error handling, and end-to-end integrity. We present the `Connector` design, describe implementations for different storage services, evaluate trade-offs inherent in managed vs. direct transfers, motivate recommended deployment options, and propose a model-based method that allows for easy characterization of performance in different contexts without exhaustive benchmarking.

CCS Concepts: • **General and reference** → *Evaluation*; • **Computing methodologies** → *Distributed computing methodologies.*

Additional Key Words and Phrases: Data Transfer, Cloud Storage, Storage Interface

1 INTRODUCTION

Easy access to data produced by scientific research is essential if such products are to be widely available for research, education, business, and other purposes [47]. Far from being mere rehashes of old datasets, evidence shows that studies based on analyses of previously published data can achieve just as much impact as the original projects [25]. Reducing barriers to the sharing of scientific data is a multi-faceted challenge [65], but one fundamental need is efficient, secure, and reliable access to data, regardless of location.

This Preprint has been accepted to appear in the ACM Transactions on Modeling and Performance Evaluation of Computing Systems.
Authors' addresses: Zhengchun Liu, Argonne National Laboratory, 9700 S. Cass Ave., Lemont, IL, 60439, zhengchun.liu@anl.gov; Rajkumar Kettimuthu, Argonne National Laboratory, kettimut@anl.gov; Joaquin Chung, Argonne National Laboratory, chungmiranda@anl.gov; Rachana Ananthakrishnan, The University of Chicago, rachana@globus.org; Michael Link, The University of Chicago, mlink@globus.org; Ian Foster, Argonne National Laboratory, The University of Chicago, foster@anl.gov.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
Manuscript submitted to ACM

The work described here is concerned with addressing two important obstacles to scientific data access, namely storage system diversity and efficient data movement. Ideally, once a scientist locates data of interest, they should be able to retrieve required components easily, reliably, efficiently, and securely, without concern for the details of the source and destination storage systems. In practice, considerations such as domain practices, cost, performance, data source, and analysis workflows result in scientists storing data on a wide range of storage systems, often with idiosyncratic interfaces: for example, commercial cloud object-based storage, such as Amazon Simple Storage Service (S3), Google Cloud Storage, and Microsoft Azure Blob Storage; community object-based storage solutions, such as Ceph and OpenStack Swift; parallel filesystems, such as Lustre, GPFS, and Intel DAOS; and cloud-based file hosting services and synchronization services, such as Box, Google Drive, Microsoft OneDrive, and Dropbox. Furthermore, the datasets that are created and analyzed in science are frequently large, reaching terabytes or even petabytes in size. And while analyzing data in place may be preferred [31], it is often not possible due to computational limitations or other factors. Thus, scientists must be able not only to access diverse data stores but to optimize data movement among different combinations of such systems. This need creates its own challenges in terms of leveraging these resources without overburdening application researchers [52].

This quest for universal data access has spurred many proposals over the years, from general solutions like Gopher [27] and HTTP [11] to more specialized approaches like WebDAV [64] and OpenDAP [20]. However, such solutions have been concerned primarily with enabling uniform data access, not efficient data movement among different systems. Important step towards enabling seamless data access and movement among diverse storage systems was taken more than a decade ago, with first the definition of the GridFTP protocol [2] and then, within the Globus implementation of the GridFTP protocol, the development of the Data Storage Interface (DSI) [3], within the Globus implementation of the GridFTP protocol, as a unified storage interface for use by data movement tools. Globus GridFTP and its DSI were initially supported only on POSIX-compliant file systems [3], but the storage landscape increasingly includes cloud object stores, tape archives, and other proprietary storage systems. Technically, there are two major challenges of extending DSI to support modern cloud storage service: (1) a module to enable the authentication process of the cloud storage and the delegation or passing of credential from user to connector either directly or through a service like Globus Auth [59]; (2) For security concern, there are restrictions on APIs for accessing cloud storage, for example one cannot call API too frequently which is not true to the DSI that interact with local storage systems. The evolution of DSI to accommodate new storage systems while maintaining backward compatibility, and also to incorporate support for third-party transfers and modern authentication and authorization mechanisms, produced the **Connector** abstraction that we describe in this paper. This abstraction, as instantiated in an interface and implementation, enables a wide variety of storage systems to be accessed in a consistent, performant, and secure manner, simply by installing the required `Connector` server software [16]. Equally important, it supports the management of transfers by cloud-hosted management services, such as the Globus service that we consider in this paper.

While a uniform interface to storage has many advantages, any abstraction layer tends to introduce overheads that can impact performance. Understanding the nature of these overheads is essential to determining where and when the abstraction may be used. To develop this understanding, we first present here the `Connector` abstraction and then evaluate overheads and performance when the Globus implementation of this abstraction is used to move data between diverse storage systems. The primary contributions of this paper are the following:

- We describe `Connector`, a data storage interface that permits uniform access to a wide range of storage systems, including both cloud storage and conventional file systems, while supporting third-party managed transfers.

- We propose a performance model-based method for exploring performance issues in different contexts without exhaustive benchmarking.
- We draw conclusions about implications for the `Connector` design and its Globus implementation, and recommend best practices.

The rest of this paper is organized as follows. In §2, we motivate the need for a uniform interface for data movement across diverse storage systems, including cloud storage services. In §3, we describe how `Connector` extends the original DSI to address new challenges in cloud-based storage service. We describe the details of `Connector` and six sample implementations in §4. In §5, we present a performance-model-based approach to studying costs associated with `Connector` data movement, which we show in §6 can be used to analyze the throughput of `Connector`-based data movement. In §7, we evaluate the influence of data integrity checking, which shows unique characteristics to `Connector`. In §8 we discuss best practices for production deployment. In §9 we review related work , and in §10 we summarize our conclusions and discuss future work.

2 MOTIVATION

The science and engineering community uses a large and growing number and variety of storage systems. Each such system has been created in response to specific needs for storing and accessing science and engineering data, needs that cover a broad spectrum of cost, scale, performance, and other requirements. Different systems focus on distinct requirements, provide distinct services to their clients, and often implement different interfaces and protocols for data access.

POSIX I/O provides open(), close(), read(), write(), and lseek() operations, with strict consistency and coherence requirements: for example, a write operation must be visible to other clients immediately after the system call returns. POSIX serves as the I/O interface for many storage systems, such as Lustre and GPFS, that are widely used in science institutions. Although high-level parallel I/O middleware libraries, such as MPI-IO [57], HDF5 [26], ADIOS [38], and PnetCDF [35], provide relaxed semantics for file system access, most scientific HPC applications use POSIX as their default interface for interacting with local storage [44].

Object stores, widely used in cloud computing, manage data as objects instead of files. They provide a single flat global name space and support just a few simple operations, such as PUT and GET, and a weaker form of consistency: eventual consistency. These features can simplify use of lower-cost commodity hardware and high-speed access. However, although APIs provided by cloud storage providers enable high-speed access to data from *within* the cloud, they are not typically configured to optimize data of movement *between* cloud and other remote locations, such as storage systems at science institutions or other cloud service providers. Furthermore, cloud APIs are not easily incorporated into research automation tools such as workflow systems and HPC schedulers, which typically use the POSIX interface for data staging.

2.1 Third-Party Transfer

Most remote access mechanisms encountered in storage systems are designed to support client-server or *two-party* transfers. That is, a user (client) wanting to read or write data first authenticates to the storage system (server) and then interacts with the server via a series of client-server interactions. However, while simple, this interaction modality has numerous limitations, due notably to the need for clients to manage various complexities of the data transfer process, such as detecting and responding to failures.

Thus an important data transfer pattern in many science and engineering settings involves a "third party" (e.g., an agent working on behalf of a user) initiating and managing data movement between two remote computers (or instruments). Support for such *third-party data transfers* allows users to initiate, manage, and monitor data movement from anywhere, without direct access to the systems involved in the data movement. Logic for detecting and responding to failures (e.g., via retries) can be incorporated into the agent, simplifying the logic required at the client. This feature also facilitates the integration of data movement operations into a wide variety of data automation tools, from shell scripts to scientific workflow engines, such as Galaxy [28], Kepler [4], Parsl [9], Pegasus [23], and Swift/T [66]. The ability to request data transfers enables the workflow systems to execute transparently on remote resources.

A third-party transfer necessarily engages two distinct communication channels, one for control and one for data. The control channel is used for sending protocol messages between system components, for example, from the third-party management service to data movers, in order to authenticate and authorize users and to initiate streaming. The data channel provides the link between the source and destination of the data being transferred. This split-channel model is beneficial when scientific data are located on specialized storage systems, accessible only via dedicated data transfer nodes (DTNs) [21] to which users do not have direct access, allowing for separation of control and data messages in this context [14].

2.2 Transfer Management

While servers connected to storage are the workhorses for data movement, the clients that drive data transfers play a critical role in determining transfer performance and reliability. The client needs to provide all parameters for any transfer, including the security credential(s) to be used, network usage levels (e.g., number of concurrent connections), and integrity and privacy levels. The configuration of these parameters by the client, as part of the request that it issues to servers to transfer files, has significant impact on transfer performance. Moreover, while the GridFTP protocol is built to support reliable transfers, the onus falls on the client to track the information sent back from the servers on how much data has been moved and to request restarts when needed to ensure a complete transfer. Similarly, any failures, including those resulting from integrity checks performed upon completion of a file transfer, are returned to the client; it is up to the client application to request that data be re-transferred in that situation. This management task is yet more complex in the case of transfers that include recursive transfers of directories, as the client then needs to expand directories and track progress at a per-file level in order to ensure that all files and folders are moved.

These difficulties provide further motivation for third-party transfer services such as Globus [15] and PhEDEX[51], as they enable clients to outsource such complexities to a third party. The cloud-hosted Globus transfer service, for example, implements a simple REST API for requesting transfers. The Globus service achieves efficiency by combining information on the files that need to be moved and the capacity at source and destination to determine performance parameters. It also provides reliability by tracking transfer progress and retrying on faults, and negotiates the security needed to navigate transfers between sites. By thus providing a fire-and-forget solution for users, it delivers significant usability benefits.

The `Connector` storage abstraction layer thus has two roles: (1) to enable efficient access to data stored on a variety of different storage systems and (2) to support the operation of managed transfer applications such as Globus so that they can achieve secure, performant, and reliable transfers across diverse combinations of such systems. The latter role requires specialized capabilities, such as enabling a third party to establish their identity and authority to make a request to the storage system; request that a transfer be initiated; enable encryption; monitor transfer progress; and detect errors and termination; request checksums.

3 A UNIFORM STORAGE INTERFACE: FROM DSI TO CONNECTOR

Allcock et al. [3]'s original DSI was designed to provide a uniform data storage interface for the various file systems and data management systems found in research institutions, including in addition to Posix, HPSS [34], Xrootd [24], and iRODS [49]. It was intended primarily for use by the open source Globus implementation of GridFTP, which when transferring data between two storage systems would use the appropriate DSI implementation at each end. A DSI implementation accepted requests such as stat, send, and recv and performed these functions by using the appropriate APIs of the storage system with which it interfaced. DSI consists of several function signatures and a set of semantics.

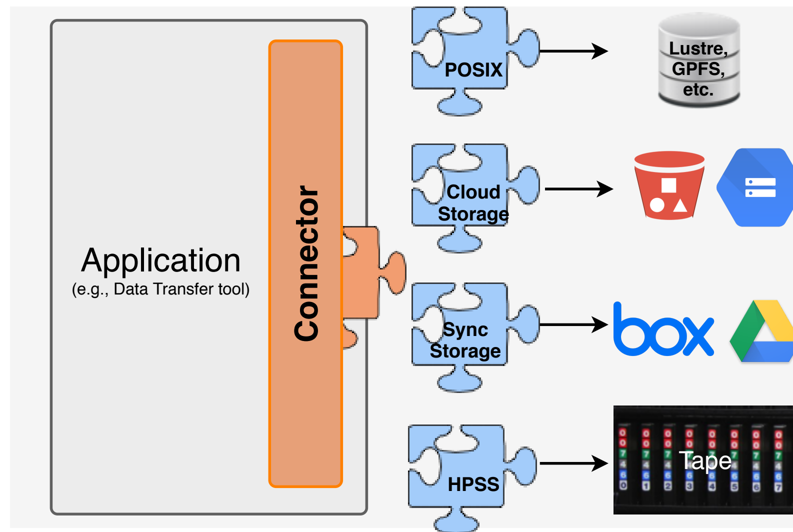


Fig. 1. The Connector abstraction.

Continued advances and diversification in network bandwidth and data storing/management techniques, such as object stores, cloud-based storage services, and conventional parallel file systems, led to the original DSI design being no longer able to handle efficiently the many different data store/retrieval APIs and authentication methods. Subsequent extensions to the original DSI to support modern authentication methods, handle certain limitations of cloud storage APIs (such as call quotas), incorporation of automatic retries and fault-tolerant capabilities, and additional management capabilities, produced what we refer to here as the `Connector` abstraction in Figure 1, as shown in Figure 2 and detailed in §4. An implementation of this abstraction is created by instantiating the various `Connector` functions, which an application program (GridFTP in our case) can then use to establish a new storage access session, read data from the storage system and send it to the application, receive data from the application and write it to the storage system, and so forth. Various of these functions themselves use callbacks (upcalls [19]) to the application program for such purposes as determining desired concurrency and buffer sizes, determining which data to send (including partial ranges due to restarts), and signalling completion of write operations so that the application can generate performance records and restart markers.

Applications can load and switch `Connectors` at runtime. When the application requires action from the storage system (e.g., store/retrieve data or metadata, directory creation), it passes a request to the loaded `Connector` module. The `Connector` then services that request and notifies the application when it is finished.

The first Connector developed by the Globus team was for POSIX-compliant file systems. Since then, researchers around the world have implemented others, some in collaboration with the Globus team and some independently. Examples include HPSS [32], iRODS [8], StoRM [55], SDQuery [56], Xrootd [24], Swift [45], and MAPFS [48].

4 GLOBUS CONNECT SERVER ARCHITECTURE AND CONNECTOR IMPLEMENTATION

Rapid growth in both the use of the Globus service and the variety of available cloud and other storage resources has created the need for additional Connectors. We introduce here six Connectors that are integrated into the Globus Connect Server (GCS) implementation [43], and for which we will provide performance data in later sections. GCS uses the Connector abstraction and interface for all interactions secure with storage systems and includes key capabilities such as security protocols needed by the storage system, management of security credentials or tokens, limit and throttling policy management, and the key IO access to the storage system. Figure 2 shows the authentication flow required to make a Connector work with Globus.

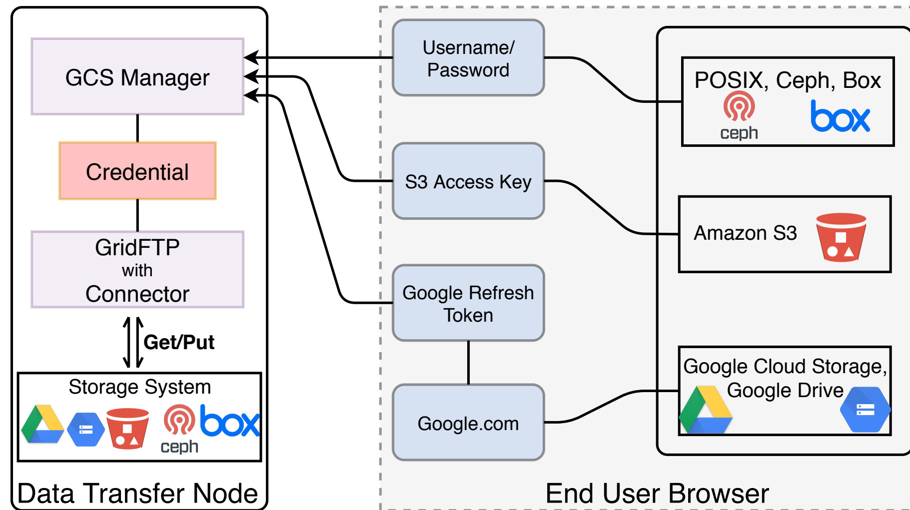


Fig. 2. Data and authentication flow of Connector.

Every storage blob requires that a credential be registered with the endpoint's GCS Manager service via its REST API. The credentials are never sent via the hosted Globus transfer service; instead, they are sent directly from the user's client (browser or command line client) to the GCS server. When the storage blob is accessed, the credential is read from the GCS Manager by the GridFTP server and passed to the Connector. For POSIX, Box, and Ceph connectors, the credential is simply the local username to which the user's login identity is mapped. For AWS S3, it is a user-submitted S3 Access Key ID and Secret Key, and for Google Drive and Google Cloud Storage, it is a token that is sent to the GCS Manager directly by Google after the user completes the Google OAuth2 login. Note that this caching of credentials is an important advantage of the GCS approach over that taken by non-persistent GridFTP clients such as globus-url-copy [3], which musst pass through credentials with each request. In the rest of this section, we briefly introduce the cloud stores to which we apply our performance modeling-based evaluation method of Connector in the next section.

Amazon Simple Storage Service (AWS-S3) is a service offered by Amazon Web Services (AWS) that provides object storage through a web service interface.

Wasabi [62], an enterprise-class, tier-free cloud storage service, provides an S3-compliant interface to use with storage applications, gateways, and other platforms.

Google-Drive is a file storage and synchronization service developed by Google. G Suite [30], a suite of cloud computing, productivity and collaboration tools, software, and products developed by Google Cloud, is widely used by education institutions, with which users have significant storage allocations within Google Drive. It also is being used as second-tier storage for research data. Connector helps in handling certain limitations of the Google Drive API (such as call quotas) through automatic retries and fault-tolerant capabilities in Globus transfer service.

Ceph [63], an open-source software storage platform, delivers object, block, and file storage in one unified system. It is based on Reliable Autonomic Distributed Object Store, which distributes objects across a cluster of storage nodes and replicates objects for fault tolerance. Ceph decouples data and metadata. Object Storage Devices store data, and Metadata Servers stores metadata, with metadata distributed dynamically among multiple Metadata Servers.

box.com provides a service similar to Google-Drive. Its growing use in universities and national laboratories for research data motivated the development of a Box Connector, which enables bridging to other storage and, as with Google Drive, handles limitations of the native API.

Google-Cloud storage, like AWS-S3, is a RESTful file storage web service for storing and accessing data on Google Cloud Platform infrastructure. Its service specifications make it more suitable than Google-Drive for enterprise use. Its growing use for research data motivated the implementation of a Google-Cloud Connector, which is being used to move data between Google-Cloud and research institute storage and between AWS-S3 and Google-Cloud.

5 PERFORMANCE MODELING-BASED OVERHEAD EVALUATION

One way to evaluate a data transfer tool is via exhaustive enumeration of many different environments and scenarios. However, this approach can easily consume a great many resources. Liu et al. observe that per-file transfer start-up costs are the performance killer when transferring many small files between science facilities [39], and that science workloads often, unfortunately, involve many small files [43]. We describe here a performance model that captures these per-file startup costs, and present experiments that allow us to measure indirectly those costs when a cloud Connector is involved in a transfer. It is important to note that both the model and experiments are designed purposely for measuring the per-file startup cost from the perspective of the data transfer tool. This cost depends only on the endpoints involved in a transfer, the network that connects those endpoints, and the transfer service: it is agnostic to file size and dataset characteristics (we conduct experiments in §5.5 to verify this claim). Thus the per-file startup costs that we measure with the proposed method are directly transferable to other transfers with different file size and dataset characteristics, as long as the same endpoints are involved. The throughput analysis in §6 confirms this statement. All experiment source code, test-beds environment setup instructions and experiment result analysis code are available at https://github.com/ramsesproject/dsi.

5.1 Colocated and Proxy Connector Scenarios

A Connector is usually colocated with the storage system to which it provides access so that it can combine efficient network communications (via GridFTP [3, 37]) with efficient storage system access (via native storage system APIs). As we will see, this *colocated connector* model is usually the most efficient means of accessing cloud storage systems. However, not all cloud storage systems allow for the deployment of a Connector immediately adjacent to cloud storage. Thus, we sometimes need to use a *proxy connector* model, which a Connector deployed at some remote location employs

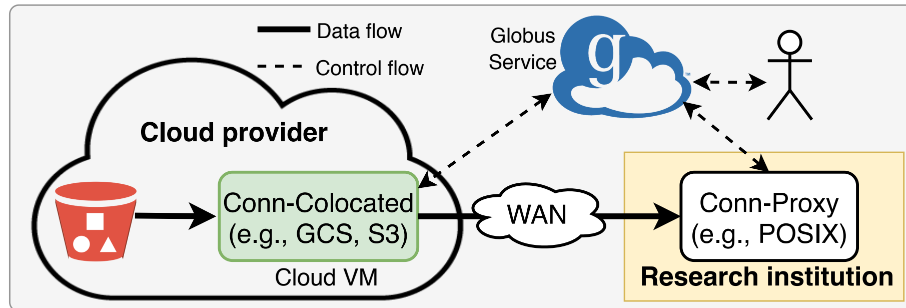


Fig. 3. The *colocated connector* scenario, in which the `Connector` (`Conn-Colocated`) that interacts with a cloud storage service is deployed in the same cloud. This scenario is used in `AWS-S3` and `Google-Cloud`.

remote access protocols provided by the cloud storage system for access. Figures 3 and 4 illustrate the colocated and proxy connector models, respectively.

In the experiments for which we report results below, we use the proxy connector model for cloud storage systems that do not permit the colocation of a `Connector` with cloud storage: `Wasabi`, `box.com` and `Ceph`. In these cases, the proxy connector uses cloud storage-provided remote access protocols to access the cloud storage. For `AWS-S3` and `Google-Cloud`, we leverage their cloud computing capabilities to deploy `Connectors` adjacent to the cloud storage (specifically, within the same cloud provider *region*). Here, cloud storage (`AWS-S3` or `Google-Cloud`) APIs are used only for local data access and GridFTP is used to move data over the wide area network. In the experiments presented below, we also compare the performance that can be achieved for `AWS-S3` or `Google-Cloud` with the proxy and colocated models.

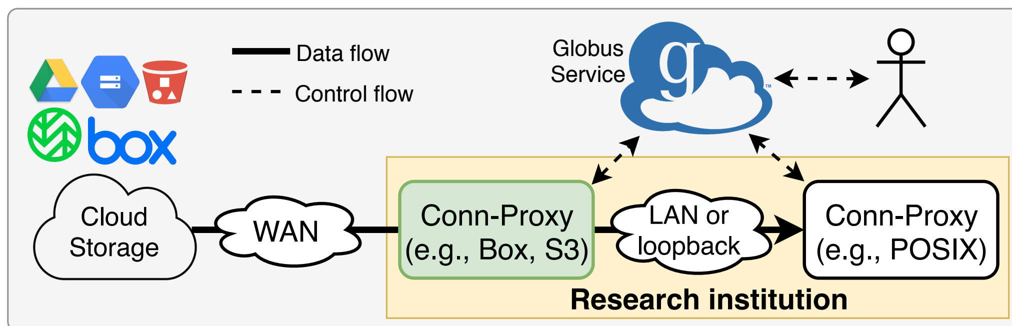


Fig. 4. The *proxy connector* scenario, in which a `Connector` (`Conn-Colocated`) that interacts with a cloud storage service is deployed remotely from the cloud provider, for example at the same location as another `Connector` (`Conn-Proxy`), to/from which transfers are to occur.

5.2 Analysis of Experiment Results

5.2.1 Regression analysis. This statistical process enables estimation of the relationship between a dependent variable (e.g., data movement performance) and one or more independent variables (e.g., file sizes). Consider the model function

$$y = \alpha + \beta x, \tag{1}$$

which describes a line with slope β and y-intercept α. This relationship may not hold exactly for the largely unobserved population of values of the independent and dependent variables; we call the unobserved deviations from this equation the *errors*. Suppose we observe n data pairs and call them (x_i, y_i), $i = 1, \dots n$. We describe the underlying relationship between y_i and x_i involving the error term ϵ_i by

$$y_i = \alpha + \beta x_i + \epsilon_i. \tag{2}$$

We then estimate α and β by solving the following minimization problem:

$$\min_{\alpha,\beta} Q(\alpha, \beta), \quad \text{for } Q(\alpha, \beta) = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2. \tag{3}$$

5.2.2 Performance model. We consider the transfer of multiple files sequentially between two endpoints. We assume that each file introduces a fixed file startup cost t_0 [39], and that the end-to-end theoretical throughput (the minimum of source read, network, and destination write throughputs, as studied by Liu et al. [40]), is R. Then the time T to transfer N files totaling B bytes when transferring the N files one by one is

$$T = N \times t_0 + \frac{B}{R} + S_0, \tag{4}$$

where S_0 is a transfer startup cost in seconds, to be measured in §5.4. S_0 will typically be lower for two-party transfers but higher for third-party transfers, due to the need in the latter context for coordination between transfer client, transfer service, and source and destination servers.

We can combine Equations 3 and 4 by setting $\alpha = \frac{B}{R} + S_0$ and $\beta = t_0$ to estimate indirectly the per-file startup cost, t_0. Since S_0 is typically a constant value, α will be inversely proportional for large B to R, the end-to-end throughput, and will thus reflect the network use `efficiency`, namely, how fast the network can transfer one single large file. A smaller α in the fitted model corresponds to a more efficient system.

5.2.3 Pearson correlation. This coefficient [10], $\rho(x, y)$, is a measure of the linear correlation between variables x and y. It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation. It is calculated by:

$$\rho(x, y) = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y}, \tag{5}$$

where $\text{cov}(x, y) = E\left[(X - \mu_x)(Y - \mu_y)\right]$ is the co-variance of variables x and y, and σ_x and σ_y denote the standard deviation of x and y, respectively.

We performed experiments to verify the model of Equation 4, by checking whether measured T and N values have a strong linear relationship, and to estimate the overhead (i.e., t_0) of a file indirectly. We then used Pearson correlation to quantify the linear relation between data transfer time, t, and number of files, f, in datasets that total to the same total size across multiple experiments. Table 1 presents the correlation coefficient between data transfer time and number of files for transfers to and from the six storage systems using `Connector` (deployed locally and at cloud if applicable) as

well as native API. We see that the coefficients are close to 1 for all cases, indicating a strong linear relation between transfer time and number of files. Thus we can use Equation 4 as a performance model for data transfers to/from all six storage systems and use regression analysis to resolve the model parameters.

In all experiments, we kept the total dataset size fixed but varied the number of files. We chose the total dataset size to keep the experiment time long enough to mitigate transfer service startup cost (i.e., per-transfer overhead) but short enough to reduce the influence of fluctuating storage and network load [41]. To this end, we used a 5 GB dataset for `Wasabi`, `AWS-S3`, and `Google-Cloud` and a 1 GB dataset for `Google-Drive` and `box.com`, considering their difference in peak end-to-end performance. Without loss of generality, we split each dataset into $N \in \{50, 100, 200, 400, 600, 800, 1000\}$ equal-sized files. It is important to note that this is a purposely designed experiment to indirectly measure the per-file overhead and transfer service efficiency.

We moved the dataset using both the appropriate `Connector` and the service providers' native APIs, optimized by the provider for reading/writing from/to the storage service, and then used the regression analysis methods described above to build a performance model for both the `Connector` and the native API. To mitigate the influence of external load on the cloud and network, we repeated each experiment until the change in average transfer time of all repetitions is less than 5%.

Table 1. Correlation coefficient, $\rho_{(t,f)}$, between transfer time t, and number of files, f, to and from different storage systems using the native API and `Connector` deployed locally (at a science institution) and at the cloud (where applicable).

Transfer Direction	Connector-Proxy	Connector-cloud	Native-API
To `AWS-S3`	0.999	0.973	0.995
From `AWS-S3`	0.989	0.993	0.989
To `Wasabi`	0.999	N/A	0.998
From `Wasabi`	0.997	N/A	0.998
To `Google-Cloud`	0.997	0.999	0.993
From `Google-Cloud`	0.999	0.996	0.992
To `Google-Drive`	0.994	N/A	0.992
From `Google-Drive`	0.989	N/A	0.995
To `Ceph`	0.996	0.999	0.999
From `Ceph`	0.986	0.994	0.976
To `box.com`	0.998	N/A	0.999
From `box.com`	0.996	N/A	0.998

5.3 Results and Discussion

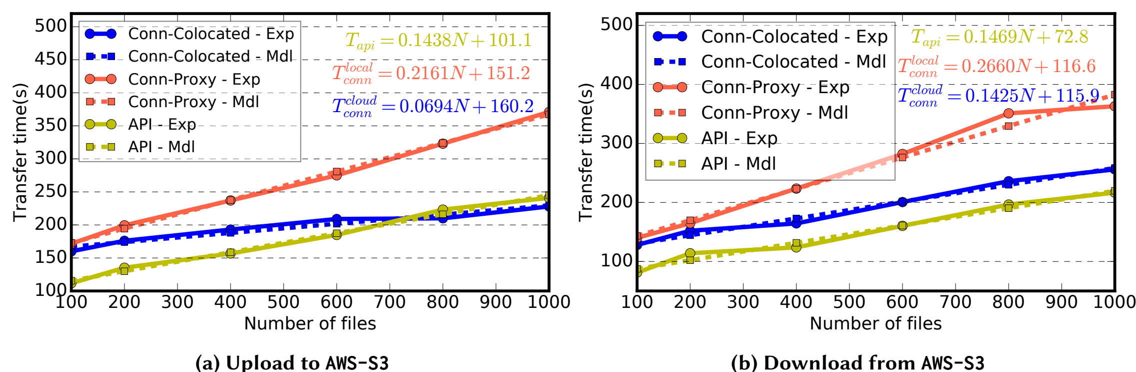
We performed the following experiments without Globus integrity checking [15, 16]. Globus integrity check detects any data corruption that occurred during transmission over the network and/or while writing data to the destination storage by reading the data at the destination (after it was written to the storage), computing the checksum, and verifying it with the source checksum. In practice, integrity checking is of vital importance for storage-to-storage transfers [17, 43, 54]. See §7 for more discussion of the importance and performance impacts of integrity checks.

We use the following abbreviations and terminology in figures throughput the paper.

- `Exp`: An experimental measurement.
- `Mdl`: A predicted value using the proposed performance model.
- `Conn`: An abbreviation for `Connector`.

- `-Colocated`: A suffix indicating a `Connector` deployed within the cloud provider that operates a cloud storage service (i.e., on AWS for S3 or Google Cloud VM for `Google-Cloud`), as illustrated in Figure 3.
- `-Proxy`: A suffix indicating a `Connector` deployed externally to the cloud provider, locally in a science institution, as illustrated in Figure 4.
- `API`: A native cloud storage application programming interface provided by a cloud service provider.

5.3.1 Amazon S3. We used the `boto3` [12] Python interface to AWS to download data from and to upload data to `AWS-S3` buckets, and compared its performance with that of `AWS-Connector`. Figure 5 shows experimental results and performance model predictions. When uploading from local (i.e., locally at science institution, the same for all experiments in this manuscript) to `AWS-S3`, `Conn-Proxy` performs worse than native API; `Conn-Colocated` has less per-file overhead but lower throughput than the native API has. Thus, `Connector` can outperform native API when transferring many small files, where per-file overhead become significant. For downloads from `AWS-S3` to a POSIX file system, the per-file overhead trend is similar to that for uploads, but `AWS-Connector` efficiency is worse when compare with native APIs, leading to worse performance when downloading large files.

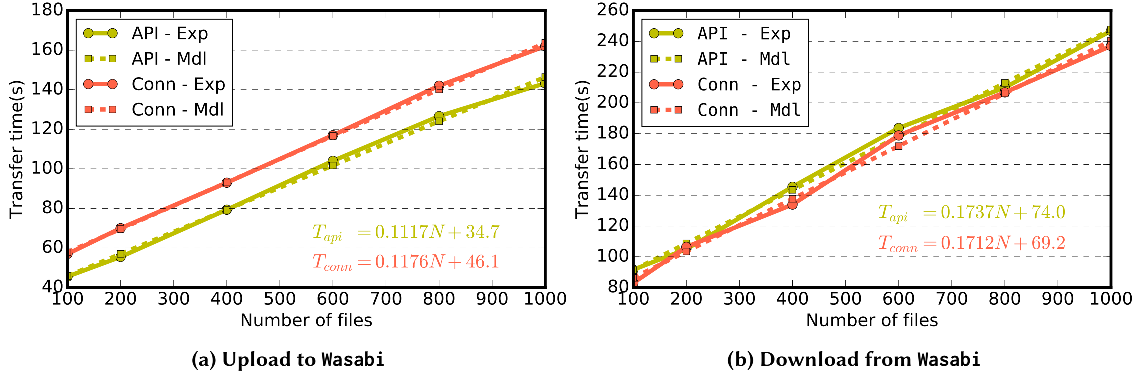


(a) Upload to `AWS-S3` **(b) Download from `AWS-S3`**

Fig. 5. Elapsed time vs. number of files for a 5 GB transfer between a POSIX file system and `AWS-S3`. `Conn-Colocated`: The `AWS-Connector` is deployed in AWS near the S3 bucket (Figure 3); `Conn-Proxy`: the `AWS-Connector` is deployed remotely from `AWS-S3`, near the POSIX `Connector` (Figure 4).

5.3.2 Wasabi. `Wasabi` provides an S3-compliant interface to use with storage applications, gateways, and other platforms. We used APIs from `boto3` (the same API used for `AWS-S3`) as the native tool to download from and upload to `Wasabi` buckets, and we compared its performance with the `Wasabi` connector.

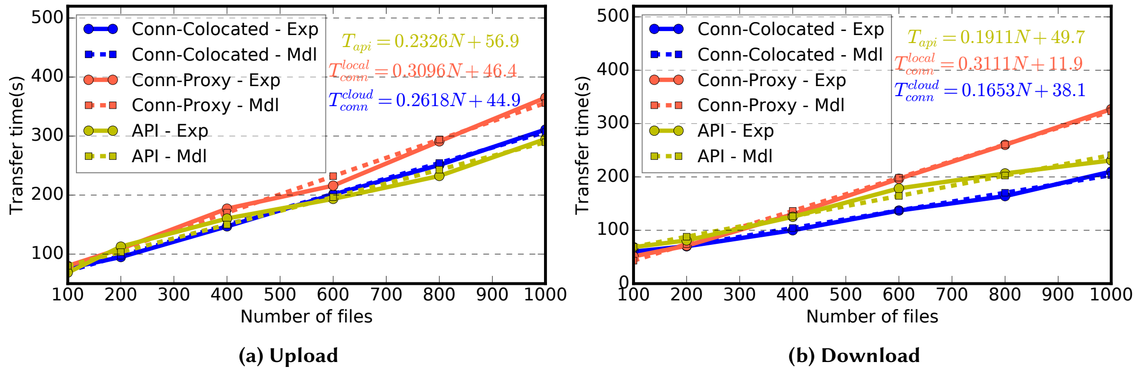
Figure 6 presents the regression analysis results for both upload to and download from `Wasabi` using `boto3` and the Globus `Wasabi` connector. We see from Figure 6 that the native tool and the `Connector` have similar per-file overheads for both download from, and upload to, `Wasabi`. In terms of average throughput achieved, the `Connector` is slightly slower for uploads and slightly faster for downloads. Overall, we conclude that the `Connector` performs worse than the native tool when uploading many large files, but is comparable to the native tool when transferring many small files, a common use case in practice. Note that this experiment was designed to measure per-file overhead, which can be mitigated by using either high concurrency or prefetching, as studied by Liu et al. [39]. We study throughput performance in §6.



(a) Upload to `Wasabi` (b) Download from `Wasabi`

Fig. 6. Elapsed time vs. number of files for a 5 GB transfer between a POSIX file system and remote `Wasabi`.

5.3.3 Google Cloud Storage. Google Cloud also provides native APIs [29] for upload to, and download from, a storage bucket. These APIs behave similarly to the `boto3` that we used for `AWS-S3` and `Wasabi`; we can authenticate once and reuse the credential to transfer all files sequentially for the regression analysis.



(a) Upload (b) Download

Fig. 7. Elapsed time vs. number of files for a 5 GB transfer between a POSIX file system and remote `Google-Cloud` storage. `Conn-Colocated`: The `Connector` used to access cloud storage is deployed in the Google Cloud near the storage bucket (Figure 3); `Conn-Proxy`: The `Connector` used to access cloud storage is deployed remotely from `Google-Cloud`, near the POSIX `Connector` (Figure 4).

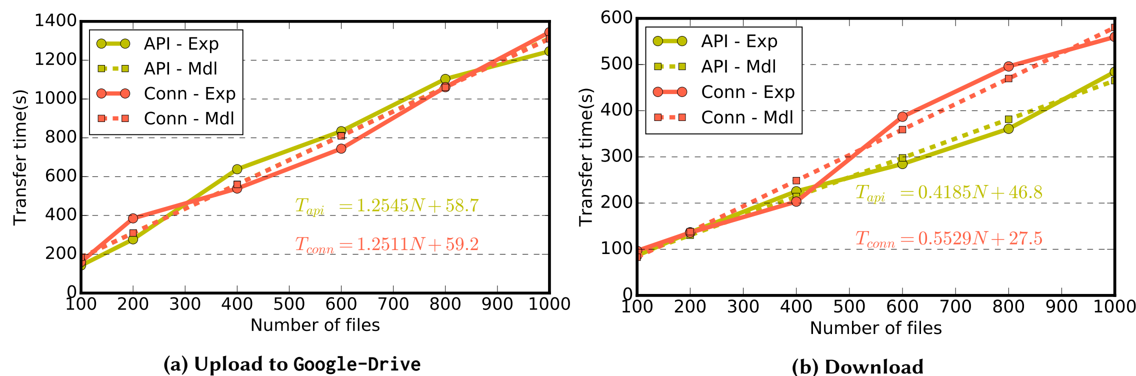
Figure 7 compares experimental results and fitted performance models. We see that a proxy deployment of `Connector` used to access cloud storage incurs much higher per-file startup costs (the first term in the equations in the figures) than when using the native API, in both data movement directions. However, as modeled in Equation 4, the `Connector` has much higher efficiency (the second term in the equations in the figures). In other words, the `Connector` performs better than the API when transferring a few big files but worse when transferring many small files.

Figure 7 also contrasts the performance achieved with the colocated and proxy deployment models. In the colocated model, when the `Connector` is deployed near the storage bucket on a Google Cloud VM (and thus GridFTP, optimized for WAN data movement, is used for WAN transfer, and the API is used only within the cloud, namely, to move data between the Cloud VM and storage bucket within the same data center), the per-file overhead of `Connector` is slightly

worse than that of the API for upload but better than the API for download, thanks to GridFTP WAN data movement optimizations [3]. Again, the `Connector` achieves much higher efficiency than does the native API. These results reveal that if the `Connector` is deployed near the storage bucket, it will perform better than the native API, if the network bandwidth is not the bottleneck.

5.3.4 Google Drive. Transfers to and from `Google-Drive` are significantly slower than for the other cloud storage services studied, achieving a transfer rate that is only about 1/6 of that achieved with `Google-Cloud`. Thus, to optimize experiment times and to minimize the influence of external load on our experiments, we used datasets totaling 1 GB, rather than 5 GB as for other `Connectors`, for our regression analysis experiments.

As `Google-Drive` does not support deployment of a `Connector` adjacent to cloud storage, our experiments (see Figure 8) compare only the proxy scenario and the `Google-Drive` API. We see that these two approaches perform similarly for uploads, while tor downloads, `GoogleDrive-Connector` introduces a little more per-file overhead than does the native APIs, but achieves higher network use efficiencies. Thus, it achieves similar performance to that of the native API for big files, but underperforms for smaller files.



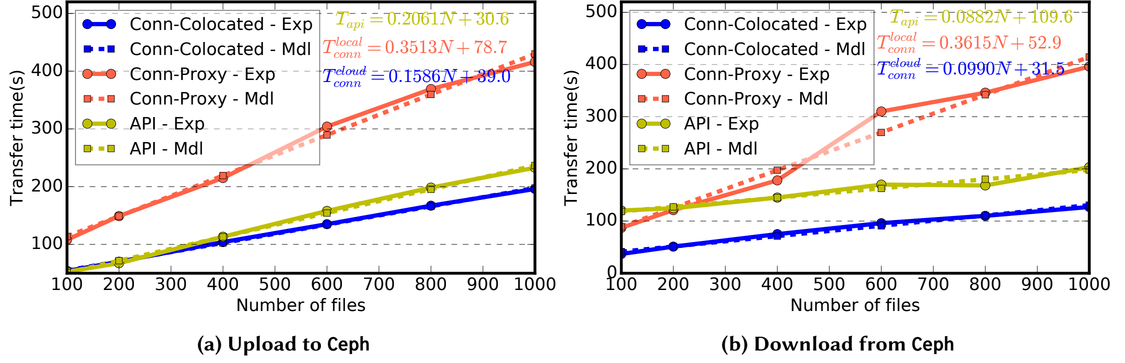
(a) Upload to `Google-Drive` **(b) Download**

Fig. 8. Elapsed time vs. number of files for a 1 GB transfer between a POSIX file system and remote `Google-Drive`

5.3.5 Ceph. Similarly to our evaluation for `AWS-Connector` and `GoogleCloud-Connector`, we consider two deployment scenarios for `Ceph-Connector`: (1) close to the Ceph storage system (referred to as `cloud`) and (2) near to the storage system from which transfers to/from Ceph are performed (referred to as `local`).

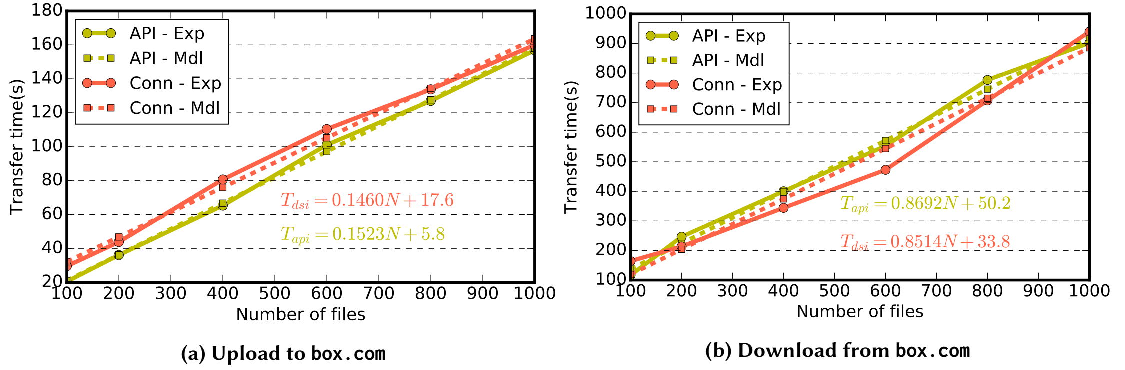
Figure 9 compares the performance model and actual experiment measurement. We see that, as in the case of `AWS-Connector` and `GoogleCloud-Connector`, the `Connector` incurs much lower per-file overheads when deployed near to Ceph. That is mostly because in that scenario, GridFTP is used for wide-area data transport, and GridFTP allows moving data out-of-order, leading to better efficiency.

5.3.6 Box.com. As we did for evaluating other `Connectors`, here again we used native APIs (here, those provided by the *Box SDK* [13]) to move data between `box.com` and local storage in order to compare Box and the `Connector`. From the experimental measurements in Figure 10, we observe that `Box.com-Connector` and the native API have similar per-file overheads.



(a) Upload to Ceph **(b) Download from Ceph**

Fig. 9. Elapsed time vs. number of files for a 5 GB transfer between a POSIX file system and remote Ceph cloud storage.



(a) Upload to `box.com` **(b) Download from `box.com`**

Fig. 10. Elapsed time vs. number of files for a 1 GB transfer between a POSIX file system and remote `box.com`

5.4 Transfer Startup Cost

Equation 4 includes, for each transfer, a startup cost of S_0. This cost varies according to the transfer method used. If a user logs in to a cloud service and initiates a two-party transfer directly, the cost may be relatively low. In the case of a cloud-hosted third-party transfer service such as Globus, it will be higher. To measure this cost in different contexts, we designed an experiment that transfers a single file of different sizes. Thus, the performance model is

$$T = B * t_u + S_0, \tag{6}$$

where B is the size of the single file in GB and t_u is the time to transfer 1 GB. To resolve S_0, we transfer a single file with $B \in \{1, 3, \ldots, 17, 19\}$ GB from a POSIX file system to a remote cloud store (in this case, `Wasabi`), and fit the resulting runtimes to Equation 6. Figure 11 shows the relation between B and T. We see a strong linear relationship between B and T and a transfer startup cost of 2.3 seconds, which is negligible in most cases except where one transfers a particularly small amount of data in a particularly high-throughput environment.

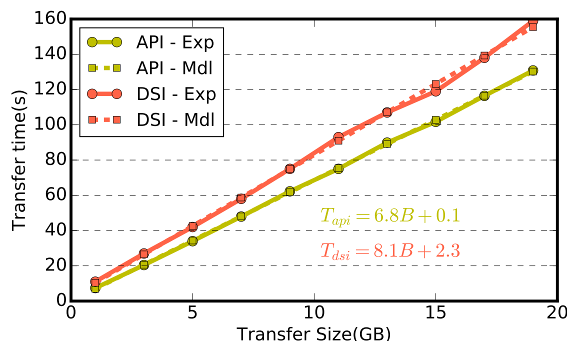
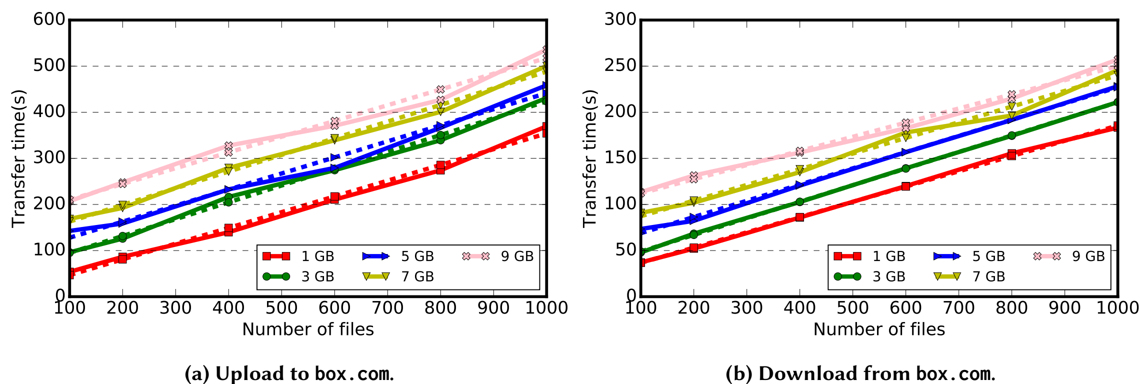


Fig. 11. Transfer time vs. single-file dataset size for upload to `Wasabi`: Globus `Connector` third-party and API two-party.

5.5 Further experiments with different transfer size

In order to prove that the proposed performance model and experiments are agnostic to transfer size, we conducted experiments in a production `Box.com-Connector` at Argonne National Laboratory (different as it used for Figure 10 which was a test-bed using a personnel `box.com` account) using different transfer size range from 1 GB to 9 GB (instead of 2GB/5GB used in the above reported experiments). Figure 12 and Table 2 showed the results. As one can see, the model is agnostic to the size of dataset used for experiments. The indirectly measured per-file startup cost remains consistent (we believe the slight variance is caused by noise because of the shared environment for experiments) in different transfer size.



(a) Upload to `box.com`. **(b) Download from `box.com`.**

Fig. 12. Elapsed time vs. number of files for different dataset size (1 GB, ..., 9 GB) transferred between a POSIX file system and `Box.com-Connector`.

6 THROUGHPUT ANALYSIS

Based on the investigation of per-file overhead, we see that datasets with big files are more friendly to transfer tools [39]. Here, we used the most friendly datasets to benchmark the best transfer performance using different concurrency levels. Specifically, in order to use a concurrency of *cc* with a `Connector`, we initiated a transfer with *cc* files, each of size 1 GB. When a native API was used, we initiated *cc* threads to transfer *cc* files concurrently. In practice, aggregated throughput

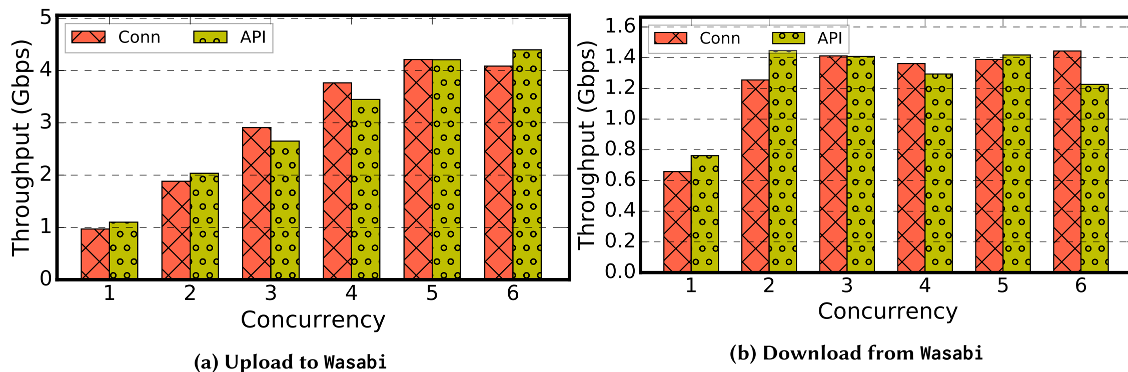
Table 2. Performance model fitted using experiment with different transfer size, for performance model verification.

Direction / Transfer-Size	Performance Model	
	Upload	*Download*
1 GB	$0.3415N + 12.62$	$0.1655N + 20.14$
3 GB	$0.3651N + 58.58$	$0.1799N + 31.00$
5 GB	$0.3484N + 93.07$	$0.1759N + 51.26$
7 GB	$0.3615N + 126.7$	$0.1699N + 70.24$
9 GB	$0.3416N + 176.2$	$0.1534N + 96.71$

first increases quickly with concurrency and eventually drops slowly, because of local contention. As noted in previous studies [7, 42, 67], there is no one-size-fits-all setting for concurrency. Thus, for all experiments in this section, we increased concurrency until we see negative benefit.

6.1 Wasabi

Figure 13 compares the S3 `Connector` and Wasabi API. We see that transferring multiple files concurrently does help to some extent by overlapping the per-file overhead. As for throughput, as evaluated in the preceding section, `Wasabi-Connector` achieves performance similar to that of the native API does.



(a) Upload to `Wasabi`

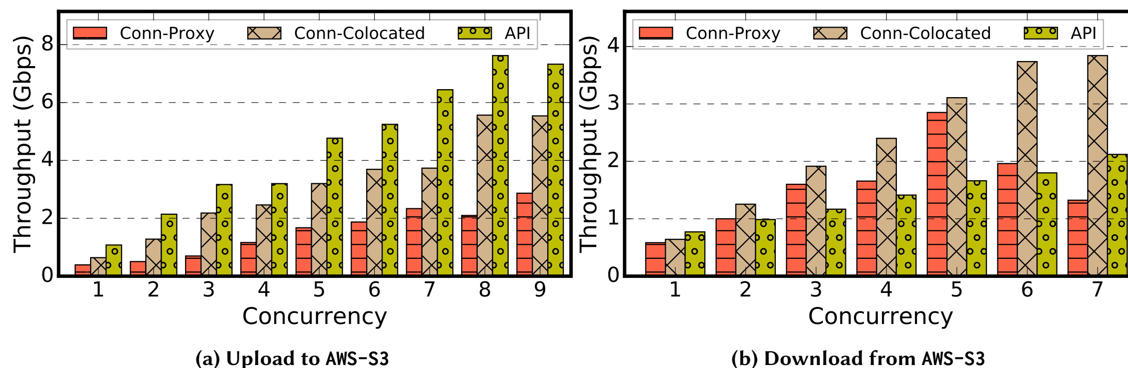
(b) Download from `Wasabi`

Fig. 13. Transfer performance as a function of concurrency: Globus `Connector` third-party and `Wasabi` two-party API

6.2 AWS S3

Figure 14 shows transfer performance between a POSIX `Connector` and AWS S3 as a function of concurrency used. We see that uploads to AWS S3 are consistently faster via the AWS API than via `AWS-Connector`, while for downloads the reverse is true. Furthermore, the `Connector` performance is consistently better when on AWS rather than local. We attribute the superior download performance of `AWS-Connector` to its use of the wide-area-network-optimized GridFTP, which for example allows out-of-order transmissions. Thus, `AWS-Connector` can extract data from S3 as fast

as S3 will allow via local area network (within AWS region) and transmit them in parallel (out-of-order if needed) over the wide area network using GridFTP.



(a) Upload to AWS-S3

(b) Download from AWS-S3

Fig. 14. Transfer performance as a function of concurrency

For downloads, the limitation seems to be the network performance of the AWS EC2 instance on which the `AWS-Connector` is located. The `m5.8xlarge` instance (32 vCPU, 128 GB RAM) that we used to host `AWS-Connector` on AWS is supposed to deliver 10 Gbps external network performance. However, an `iperf` test with 16 parallel TCP streams from the AWS instance to a DTN associated with the POSIX `Connector` showed only 4.7 Gbps (i.e., downloads in Figure 14b).

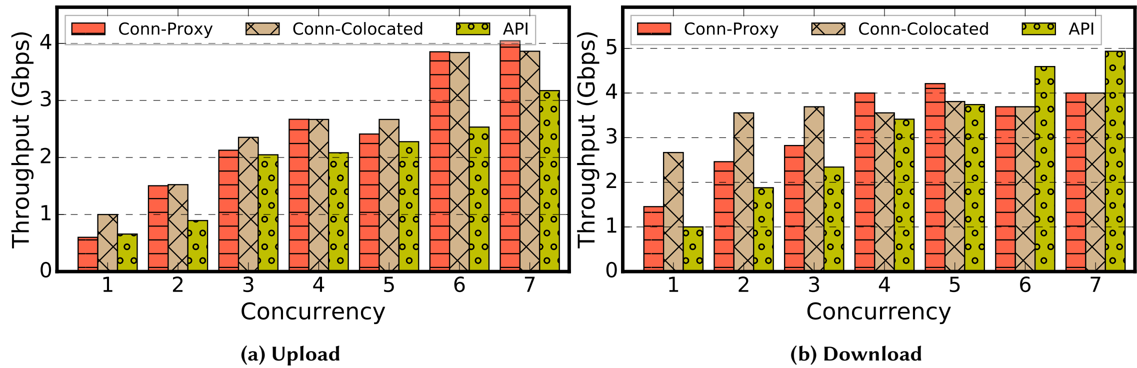
6.3 Google Cloud Storage

Figure 15 shows transfer performance as a function of concurrency used. In the `Conn-Colocated` case, the `Connector` runs on a Google Cloud virtual machine instance with 32 vCPU and 128GB RAM that is close to the `Google-Cloud` bucket. We used `iperf` with 16 parallel TCP streams to measure network bandwidth between the POSIX `Connector` DTN and the VM instance on Google Cloud; we achieved 4 Gbps from Google Cloud to the DTN (i.e., download) and 7.3 Gbps from the DTN to the Google Cloud instance (i.e., upload). Since the data will not go through this VM instance when using the native API, native API transfers are not limited by the above mentioned peak `iperf` throughput values (which are likely limited by the VM's network). Thus, it is not fair to compare the throughput achieved by the API with that of the throughput achieved by `Connector` when the throughput achieved by the API is above the peak `iperf` measurements. We see in Figure 15a that the `Connector` upload performance is consistently better than that of the native API.

In the download case, since the VM's network egress bandwidth is only 4 Gbps, the comparison after achieving 4 Gbps (there are protocol overheads in practice) does not make sense. However, in those experiment that are not limited by network bandwidth (i.e., when concurrency is less than 5), the `Connector` clearly performs better than API, with the cloud-placed `Connector` (`Conn-Colocated`) performing better than the locally placed connector (`Conn-Proxy`). These results are in line with our regression analysis in §5.3.3.

6.4 Ceph

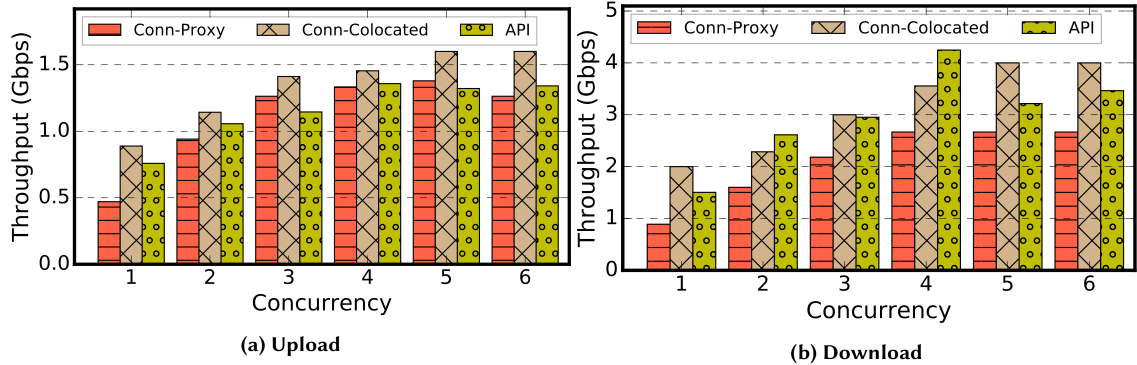
Depending on context, the `Ceph-Connector` may be deployed in either colocated or proxy mode. Here we conduct experiments to measure `Ceph-Connector` throughput and to compare with native APIs. We work with a Ceph system



(a) Upload **(b) Download**

Fig. 15. Transfer (upload to and download from `Google-Cloud`) performance as a function of concurrency

deployed on a bare metal node at the University of Chicago site of the NSF Chameleon cloud [36], and experimented with two `Ceph-Connectors` in two locations: one adjacent to the Ceph storage in Chicago and one at the Texas Advanced Computing Center (TACC). Since the data channel of `Ceph-Connector` uses the S3 protocol, we compared it against using `boto3` to access Ceph. Figure 16 compares the performance of the two `Ceph-Connector` deployments with that of the native API (i.e., `boto3`). `Ceph-Connector` always get the best performance when deployed near the Ceph system, thanks to the optimized data movement over WAN delivered by GridFTP[3].



(a) Upload **(b) Download**

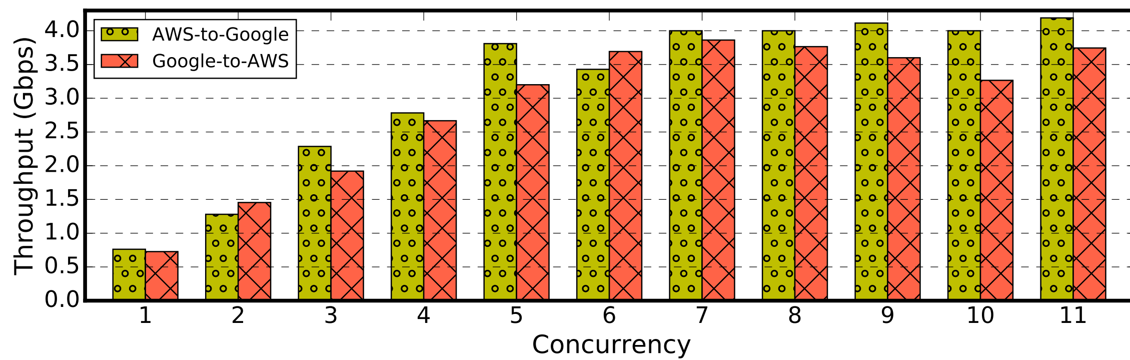
Fig. 16. Transfer (upload to and download from `Ceph`) performance as a function of concurrency

6.5 Inter-cloud Transfers

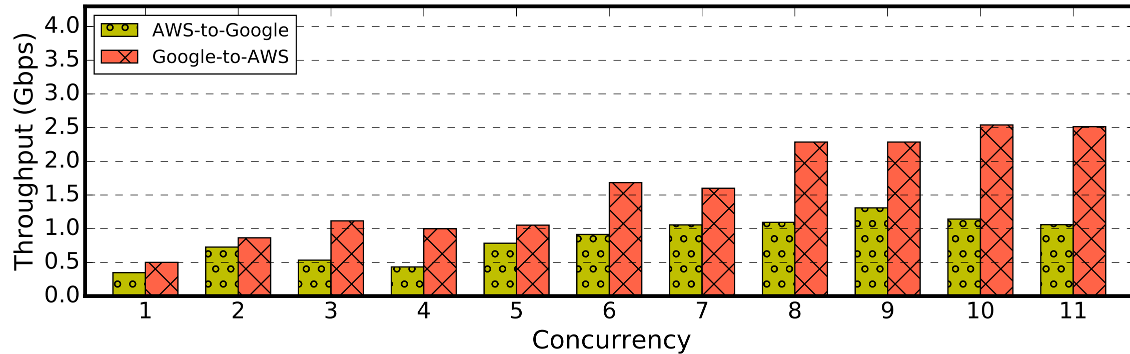
The ability to transfer files directly from one cloud store to another, instead of downloading and re-uploading files to and from an intermediate point, such as a user's workstation, can be a major boost to researcher productivity. In addition to increasing performance, the fire-and-forget nature of third-party transfer increases reliability and eliminates the need to maintain an intermediate node running for the duration of the transfer from one cloud to another.

6.5.1 Connector cross-cloud performance. Globus logs show that moving data between `AWS-S3` and `Google-Cloud` is a common use case. We first evaluated performance for moving data between cloud providers. Figure 17 shows

performance vs. concurrency when moving data between AWS-S3 and Google-Cloud using Connector. Since there is no straightforward or automated way to do cross-cloud transfers using the native cloud storage APIs, we benchmark the performance of Connector alone to determine best practice for cross-cloud transfers (more on best practices in §8). An iperf3 network speed test with 16 parallel TCP streams between the AWS VM and Google Cloud VM achieved about 4.5 Gbps in each direction. Thus, as shown in Figure 17a, Connector can reach peak throughput when the Connector is deployed at the cloud provider (i.e., one Connector on AWS to access – read or write depends on transfer direction – AWS-S3 and the other on Google Cloud to access Google Cloud Storage service). If deployed locally, however, they achieve only about half of the performance, a reduction that we attribute to network connectivity among AWS, Google Cloud, and a POSIX Connector at a science institution.



(a) Colocated connector: Connector is hosted in the cloud provider.



(b) Proxy connector: Connector is operated remote from the cloud provider.

Fig. 17. Transfer performance between AWS-S3 and Google-Cloud vs. concurrency, for (a) colocated and (b) proxy connector models.

6.5.2 Transfer agent comparisons: Globus vs. MultCloud. We also compare the cloud-to-cloud performance achieved when using Connector (via the Globus managed transfer service) with that achieved by MultCloud [46], another service that allows users to request managed third-party transfers between different cloud storage services. We show results in

Figure 18 for transfers between Google-Drive, box.com and AWS-S3. In comparing performance, we used our analysis of file size characteristics [43] to select a test dataset of 50 files totaling 1 GB. Since the free trial version of MultCloud only supports transferring files one by one, we also set concurrency to one for the Globus Connectors used for the experiment. We run both Connector on nodes provided by the Argonne-located Chameleon system, and thus in the Globus case, all data transferred from cloud A to cloud B passes via Argonne. (Cloud-based Connectors would give better performance.) We see in Figure 18 that the Connector outperforms MultCloud in all cases.

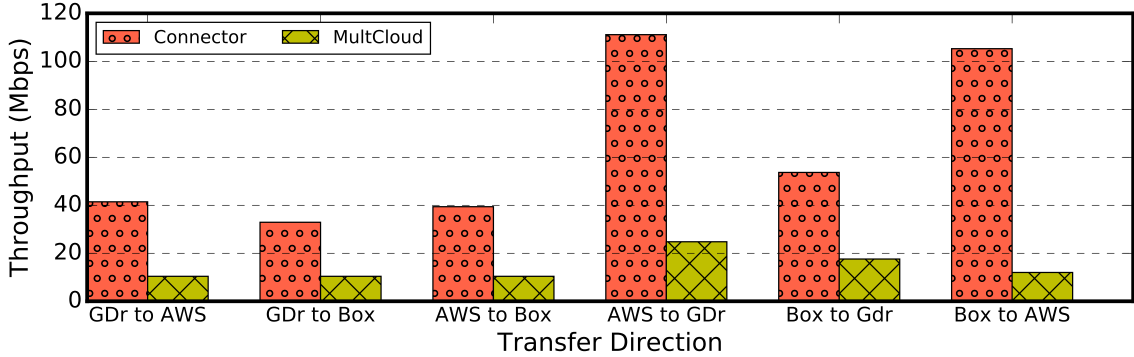


Fig. 18. Throughput comparison: MultCloud vs. Globus

7 INTEGRITY CHECKING

It is good practice to perform integrity checking on files transferred over wide area networks because factors such as faulty routers and file systems can cause silent data corruption [17, 43, 54]. The 16-bit TCP checksum is inadequate to catch network transmission errors, and other errors can occur when accessing storage. Indeed, a recent study [43] reported at least one checksum failure per 1.26 TB moved from storage to storage over a wide area network. While this number is likely an over-estimate, as it does not distinguish between data corruption and cases in which a file is modified while a transfer is in progress, it emphasizes the importance of integrity checking.

The Connector abstraction interface supports transmission integrity checking via GridFTP [3], which allows a client to request that a file be read, and a checksum computed, at the source before transmission, and then reread and a second checksum computed at the destination. The re-read action is not required when the cloud service associates a checksum/MD5 with the file/object, for example when the file size is less than a certain value. This “strong integrity checking” approach has the advantage that it can detect not only errors incurred during transport over the network but also errors incurred while writing data. However, the additional read operations can impact performance, particularly if a Connector is located remotely from cloud storage. Given the wide variety of storage systems, Connector placement strategies, and transfer workloads, we cannot provide a complete analysis of integrity checking costs. However, we present some relevant results for high throughout storage systems (where even a small integrity checking overhead can have a significant influence) in Figures 19–21, for Wasabi, AWS-S3, and Google-Cloud respectively. In each case, the Connector is located on a computer at Argonne, and the transfer involves c 300 MB files, where c is the concurrency. We see that transfer rates are lower when integrity checking is enabled, but not remarkably so, given that the file is being reread over the wide area network after writing. We attribute this result to the fact that sufficient concurrency can mitigate the overhead of integrity-checking when the DTN’s CPU is not the bottleneck.

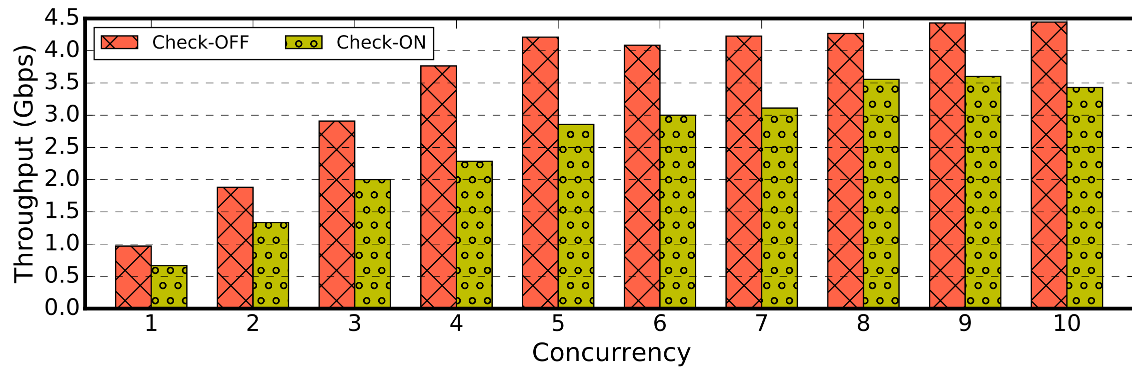


Fig. 19. Transfer (upload to `Wasabi`) performance, with integrity checking ON versus OFF

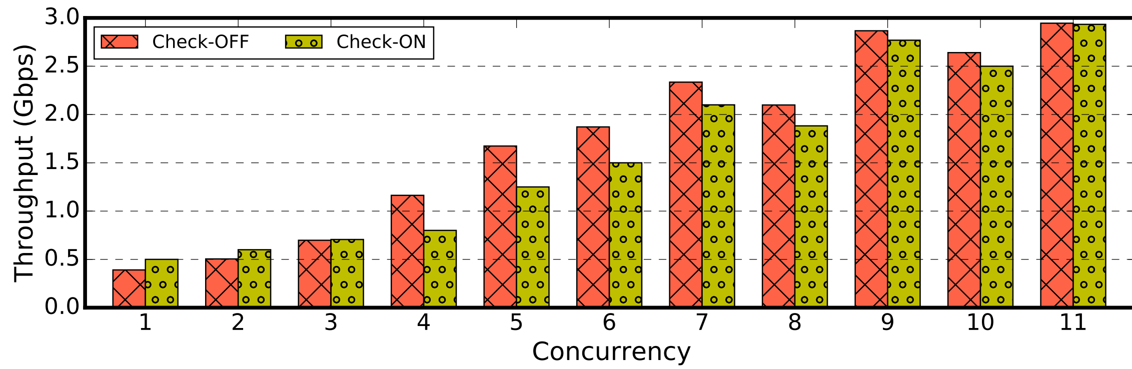


Fig. 20. Transfer (upload to `AWS-S3`) performance, with integrity checking ON versus OFF

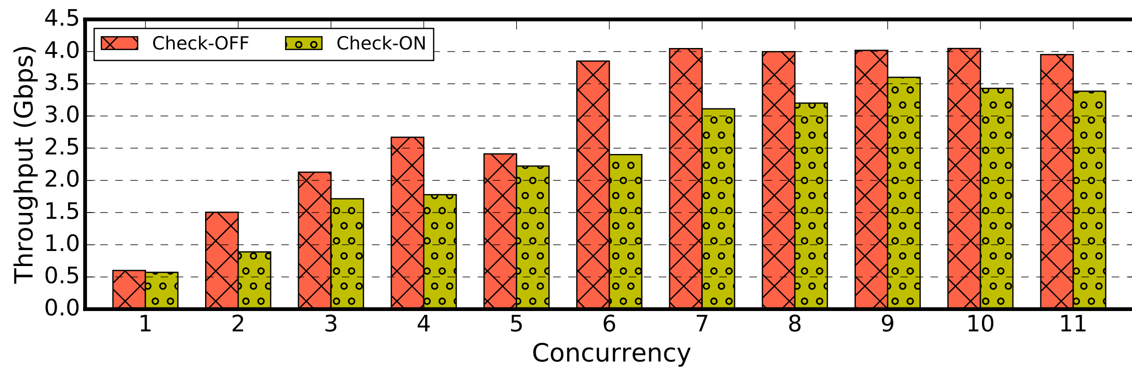


Fig. 21. Transfer (upload to `Google-Cloud`) performance, with integrity checking ON versus OFF

8 BEST PRACTICE

The GridFTP-based Globus transfer service has been heavily optimized for moving data over wide area networks [3, 37]. Here we provide recommendations for `Connector` deployment when aiming either to maximize throughput or to minimize costs.

8.1 Throughput Maximization

Best practice for throughput maximization when moving data to and from cloud storage is to deploy the corresponding `Connector` near the cloud storage service. This means, for example, using a Google Cloud computing instance as a DTN to run one or more `GoogleCloud-Connector` and using AWS EC2 instance(s) to run one or more `AWS-Connector`. Moreover, the results reported in §6.5.1 show that for inter-cloud transfers, this configuration (deploying `Connector` near the cloud storage) can achieve a 100% improvement in throughput compared to the configuration in which `Connector` is deployed locally at the users' site–or at a location that is not closer to the cloud storage. The transfer throughput achievable in these two cases depends on the size (in terms of vCPUs and memory) of the allocated instance(s). We have found that two vCPUs and 4 GB memory are needed to saturate a 10 Gbps network. In order to achieve high performance with reduced cost, these cloud-hosted DTN instances can adopt an elastic resource allocation approach, increasing resources allocated to the `Connector` when demand is high and reducing it at other times [18]. Such cloud-hosted `Connector` can be shared by several science institutions that use the same federated authentication mechanism, such as XSEDE [58].

8.2 Cost Minimization

An alternative deployment approach is to run `Connector` on computers hosted at science institutions. This approach does not require any additional hardware, but it means that all accesses to cloud storage involve data transfers with cloud provider protocols. The results presented earlier in this paper suggest that this approach will lead to little performance loss for datasets with large files but significant performance loss for datasets with many small files.

Performance-cost calculations may be different when using integrity checking, as discussed in the next section. Since `Connector` integrity checking involves rereading a file after writing and since cloud storage providers usually charge for network usage when data is moved out of the cloud, it is advantageous when integrity checking is enabled to deploy a cloud storage `Connector` in the same cloud as the storage.

9 RELATED WORK

Others have developed implementations of the Globus GridFTP DSI. EUDAT [60] implemented a DSI [8] for the Integrated Rule-Oriented Data System (iRODS) [49] data management software. Sánchez et al. [53] proposed a parallel DSI for GridFTP and offered an implementation for the MAPFS [48] parallel file system. A DSI implementation for OpenStack Object Storage (Swift) is also available [45]. However, no DSI implementation targets cloud stores, and none provide performance evaluations.

Others have developed uniform interfaces to cloud storage, but by supporting multiple protocols in a client, not a `Connector` as proposed here. We described MultCloud in §6.5.2. Rclone [50] is a command line program that offers a rsync-like tool to synchronize files for cloud storage. It integrates APIs for various cloud stores but does not provide transfer management functionality. iRODS implements an interface to AWS S3 [61].

Abramson et al. [1] proposed the Metropolitan Data Caching Infrastructure (MeDiCI) architecture to simplify data movement between different clouds and a centralized storage site. It is similar to the scenario that we evaluated in §6.5.1, but MeDiCI is cache-based, targeting on-demand cloud computing.

Liu et al. [39] used regression analysis to measure per-file overhead indirectly, and concluded that the bottleneck in transferring many small files between HPC facilities is not any single subsystem but rather the per-file overheads introduced by the major components in wide area file transfers. Deelman et al. [22] have developed similar models. The benefits of parallel streams for transfer performance are well known [33]. Several researchers have studied the impact of concurrency, parallelism, and other parameters on GridFTP transfer performance [5–7, 42, 67], for example, based on historical data [6] or lightweight probing [5].

10 CONCLUSION

We have described and evaluated the architecture and implementation of a unified storage system `Connector` designed to permit efficient, reliable, and secure integration of diverse storage systems into a distributed data movement fabric. By enabling both effective use of third-party transfers and easy plug-and-play integration of different storage system types, `Connector` simplifies both the use of those systems and the development of new `Connectors`. When used in conjunction with a managed transfer service such as Globus, the `Connectors` abstraction enables data movement across various storage systems in a "fire-and-forget" fashion.

We described `Connector` implementations for a range of storage types, from POSIX file systems to HPC parallel file systems and cloud object stores. We used a performance-model-based analysis to evaluate `Connector` implementations, and employed both experiments and analysis to elucidate implications for design and implementation. We conclude that the `Connector` model enables effective use of distributed storage with either just modest performance losses relative to native APIs, or useful performance improvements, due to the optimization of data movement over wide area networks delivered by the open source GridFTP.

The methods and results presented here have broad applicability to data movement services such as Globus and PhEDEx that suport managed third-party transfers. We demonstrate that a performant, uniform `Connector` interface can be used to support such services across diverse storage systems, including the growing cloud-based storage service. Our analysis of the costs associated with colocated vs. proxy `Connectors` emphasizes the importance of careful design of external access interfaces for cloud storage.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

[1] David Abramson, Jake Carroll, Chao Jin, Michael Mallon, Zane van Iperen, Hoang Nguyen, Allan McRae, and Liang Ming. 2019. A Cache-Based Data Movement Infrastructure for On-demand Scientific Cloud Computing. In *Supercomputing Frontiers*, David Abramson and Bronis R. de Supinski (Eds.). Springer International Publishing, Cham, 38–56.

[2] William Allcock. 2003. GridFTP: Protocol extensions to FTP for the Grid. http://www.ggf.org/documents/GFD.20.pdf.

[3] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus Striped GridFTP Framework and Server. In *ACM/IEEE Conference on Supercomputing (SC '05)*. IEEE Computer Society, Washington, DC, USA, 54–. https://doi.org/10.1109/SC.2005.72

[4] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. 2004. Kepler: An extensible system for design and execution of scientific workflows. In *16th International Conference on Scientific and Statistical Database Management*. ACM, New York, NY, 423–424. https://doi.org/10.

1109/SSDM.2004.1311241

[5] Engin Arslan, Kemal Guner, and Tevfik Kosar. 2016. HARP: predictive transfer optimization based on historical analysis and real-time probing. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, New York, NY, 288–299.

[6] Engin Arslan and Tevfik Kosar. 2018. High-speed transfer optimization based on historical analysis and real-time tuning. *IEEE Transactions on Parallel and Distributed Systems* 29, 6 (2018), 1303–1316.

[7] Engin Arslan, Bahadir A Pehlivan, and Tevfik Kosar. 2018. Big data transfer optimization through adaptive parameter tuning. *J. Parallel and Distrib. Comput.* 120 (2018), 89–100.

[8] B2Stage GridFTP [n.d.]. B2STAGE GridFTP (iRODS-DSI). https://github.com/EUDAT-B2STAGE/B2STAGE-GridFTP. Accessed April 1, 2020.

[9] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Luksaz Lacinski, Ryan Chard, Justin M. Wozniak, Ian Foster, Michael Wilde, and Kyle Chard. 2019. Parsl: Pervasive Parallel Programming in Python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, New York, NY. https://doi.org/10.1145/3307681.3325400 babuji19parsl.pdf.

[10] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise Reduction in Speech Processing*. Springer, 1–4.

[11] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. 1994. The world-wide web. *Commun. ACM* 37, 8 (1994), 76–82.

[12] Boto3 [n.d.]. AWS SDK for Python (Boto3). https://aws.amazon.com/sdk-for-python. Accessed April 1, 2020.

[13] Box SDK [n.d.]. Introducing the Box SDK. http://opensource.box.com/box-python-sdk. Accessed April 1, 2020.

[14] Kyle Chard, Eli Dart, Ian Foster, David Shifflett, Steven Tuecke, and Jason Williams. 2018. The Modern Research Data Portal: A design pattern for networked, data-intensive science. *PeerJ Computer Science* 4 (2018), e144.

[15] Kyle Chard, Ian Foster, and Steven Tuecke. 2017. Globus: Research Data Management as Service and Platform. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact* (New Orleans, LA, USA) *(PEARC17)*. Association for Computing Machinery, New York, NY, USA, Article 26, 5 pages. https://doi.org/10.1145/3093338.3093367

[16] Kyle Chard, Steven Tuecke, and Ian Foster. 2016. Globus: Recent enhancements and future plans. In *XSEDE16 Conference on Diversity, Big Data, and Science at Scale*. ACM, 27.

[17] Batyr Charyyev, Ahmed Alhussen, Hemanta Sapkota, Eric Pouyoul, Mehmet H Gunes, and Engin Arslan. 2019. Towards securing data transfers against silent data corruption. In *IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing*.

[18] Joaquin Chung, Zhengchun Liu, Rajkumar Kettimuthu, and Ian Foster. 2019. Toward an Elastic Data Transfer Infrastructure. In *15th International Conference on eScience*. 262–265. https://doi.org/10.1109/eScience.2019.00036

[19] David D Clark. 1985. The structuring of systems using upcalls. In *10th ACM Symposium on Operating Systems Principles*. 171–180.

[20] Peter Cornillon, James Gallagher, and Tom Sgouros. 2003. OPeNDAP: Accessing data in a distributed, heterogeneous environment. *Data Science Journal* 2 (2003), 164–174.

[21] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. 2014. The Science DMZ: A network design pattern for data-intensive science. *Scientific Programming* 22, 2 (2014), 173–185.

[22] Ewa Deelman, Christopher Carothers, Anirban Mandal, Brian Tierney, Jeffrey S Vetter, Ilya Baldin, Claris Castillo, Gideon Juve, Dariusz Król, Vickie Lynch, Ben Mayer, Jeremy Meredith, Thomas Proffen, Paul Ruth, and Rafael Ferreira da Silva. 2017. PANORAMA: An approach to performance modeling and diagnosis of extreme-scale workflows. *International Journal of High Performance Computing Applications* 31, 1 (2017), 4–18.

[23] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems* 46 (2015), 17–35.

[24] Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. 2005. XROOTD–A Highly scalable architecture for data access. *WSEAS Transactions on Computers* 1, 4.3 (2005), 348–353.

[25] Editorial. 2018. Data sharing and the future of science. *Nature Communications* 9, 1 (19 Jul 2018), 2817. https://doi.org/10.1038/s41467-018-05227-z

[26] Mike Folk, Albert Cheng, and Kim Yates. 1999. HDF5: A file format and I/O library for high performance computing applications. In *Supercomputing*, Vol. 99. 5–33.

[27] Philip L Frana. 2004. Before the web there was Gopher. *IEEE Annals of the History of Computing* 26, 1 (2004), 20–41.

[28] Jeremy Goecks, Anton Nekrutenko, James Taylor, and Galaxy Team. 2010. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology* 11, 8 (2010), R86.

[29] Google [n.d.]. Cloud Application Programming Interface. https://cloud.google.com/apis Accessed June 1, 2020.

[30] Google. [n.d.]. G Suit. https://gsuite.google.com Accessed June 1, 2020.

[31] Jim Gray. 2004. Distributed computing economics. In *Computer Systems*. Springer, 93–101.

[32] GridFTP-DSI-for-HPSS [n.d.]. GridFTP DSI for HPSS. https://github.com/JasonAlt/GridFTP-DSI-for-HPSS. Accessed April 1, 2020.

[33] Thomas J Hacker, Brian D Noble, and Brian D Athey. 2004. Improving throughput and maintaining fairness using parallel TCP. In *IEEE INFOCOM 2004*, Vol. 4. IEEE, 2480–2489.

[34] HPSS Collaboration [n.d.]. High Performance Storage System. http://www.hpss-collaboration.org/ Accessed June 1, 2020.

[35] Jianwei Li, Wei-keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. 2003. Parallel netCDF: A High-Performance Scientific I/O Interface. In *ACM/IEEE Conference on Supercomputing*. 39–39. https://doi.org/10.1109/SC.2003.10053

[36] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2019. Chameleon: A Scalable Production Testbed for Computer Science Research. In *Contemporary High Performance Computing: From Petascale toward Exascale* (1 ed.), Jeffrey Vetter (Ed.). Chapman & Hall/CRC Computational Science, Vol. 3. CRC Press, Boca Raton, FL, Chapter 5, 123–148.

[37] Rajkumar Kettimuthu, Zhengchun Liu, David Wheeler, Ian Foster, Katrin Heitmann, and Franck Cappello. 2018. Transferring a petabyte in a day. *Future Generation Computer Systems* 88 (2018), 191–198. https://doi.org/10.1016/j.future.2018.05.051

[38] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, Manish Parashar, Nagiza Samatova, Karsten Schwan, Arie Shoshani, Matthew Wolf, Kesheng Wu, and Weikuan Yu. 2014. Hello ADIOS: The challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 26, 7 (2014), 1453–1473.

[39] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. 2019. Data Transfer between Scientific Facilities - Bottleneck Analysis, Insights and Optimizations. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 122–131. https://doi.org/10.1109/CCGRID.2019.00023

[40] Zhengchun Liu, Prasanna Balaprakash, Rajkumar Kettimuthu, and Ian Foster. 2017. Explaining Wide Area Data Transfer Performance. In *26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington, DC, USA) *(HPDC'17)*. ACM, New York, NY, USA, 167–178. https://doi.org/10.1145/3078597.3078605

[41] Zhengchun Liu, Rajkumar Kettimuthu, Prasanna Balaprakash, Nageswara S. V. Rao, and Ian Foster. 2019. Building a Wide-Area File Transfer Performance Predictor: An Empirical Study. In *Machine Learning for Networking*, Éric Renault, Paul Mühlethaler, and Selma Boumerdassi (Eds.). Springer International Publishing, Cham, 56–78.

[42] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Peter H. Beckman. 2018. Toward a smart data transfer node. *Future Generation Computer Systems* 89 (2018), 10–18. https://doi.org/10.1016/j.future.2018.06.033

[43] Zhengchun Liu, Rajkumar Kettimuthu, Ian Foster, and Nageswara S. V. Rao. 2018. Cross-geography Scientific Data Transferring Trends and Behavior. In *27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC'18)*. ACM, New York, NY, USA, 267–278. https://doi.org/10.1145/3208040.3208053

[44] Zhengchun Liu, Ryan Lewis, Rajkumar Kettimuthu, Kevin Harms, Philip Carns, Nageswara Rao, Ian Foster, and Michael Papka. 2020. Characterization and Identification of HPC Applications at a Leadership Computing Facility. In *34th ACM International Conference on Supercomputing*. https://doi.org/10.1145/3392717.3392774

[45] Richard Moore. 2013. Data Services for Campus Researchers. https://bit.ly/2XYGKbK.

[46] MultCloud [n.d.]. Multiple Cloud Storage Manager. https://www.multcloud.com/ Accessed June 1, 2020.

[47] Irene V Pasquetto, Bernadette M Randles, and Christine L Borgman. 2017. On the reuse of scientific data. *Data Science Journal* (2017). https://doi.org/10.5334/dsj-2017-008

[48] María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. 2006. MAPFS: A flexible multiagent parallel file system for clusters. *Future Generation Computer Systems* 22, 5 (2006), 620–632. https://doi.org/10.1016/j.future.2005.09.006

[49] Arcot Rajasekar, Reagan Moore, Chien-yi Hou, Christopher A Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, Chien-Yi Hou, Christopher A. Lee, Richard Marciano, Paul Tooby, Antoine de Torcy, and Bing Zhu. 2010. iRODS primer: Integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2, 1 (2010), 1–143.

[50] Rclone [n.d.]. Rclone - rsync for cloud storage. https://rclone.org/ Accessed June 1, 2020.

[51] J Rehn, T Barrass, D Bonacorsi, J Hernandez, I Semeniouk, L Tuura, and Y Wu. 2006. PhEDEx high-throughput data transfer management system. In *Computing in High Energy and Nuclear Physics (CHEP)*, Vol. 2006.

[52] Robert Ross, Lee Ward, Philip Carns, Gary Grider, Scott Klasky, Quincey Koziol, Glenn K Lockwood, Kathryn Mohror, Bradley Settlemyer, and Matthew Wolf. 2019. *Storage systems and I/O: Organizing, storing, and accessing data for scientific discovery.* Technical Report. US-DOE Office of Science.

[53] Alberto Sánchez, María S Pérez, Pierre Gueant, Jesús Montes, and Pilar Herrero. 2006. A parallel data storage interface to GridFTP. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*. Springer, 1203–1212.

[54] Jonathan Stone and Craig Partridge. 2000. When the CRC and TCP checksum disagree. *ACM SIGCOMM Computer Communication Review* 30, 4 (2000), 309–319.

[55] StoRM GridFTP DSI [n.d.]. StoRM GridFTP DSI. https://github.com/italiangrid/storm-gridftp-dsi. Accessed April 1, 2020.

[56] Yu Su, Yi Wang, Gagan Agrawal, and Rajkumar Kettimuthu. 2013. SDQuery DSI: Integrating data management support with a wide area data transfer protocol. In *International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–12.

[57] Rajeev Thakur, William Gropp, and Ewing Lusk. 1999. On implementing MPI-IO portably and with high performance. In *6th Workshop on I/O in Parallel and Distributed Systems*. 23–32.

[58] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, Ralph Roskies, J. Ray Scott, and Nancy Wilkins-Diehr. 2014. XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering* 16, 5 (2014), 62–74.

[59] Steven Tuecke, Rachana Ananthakrishnan, Kyle Chard, Mattias Lidman, Brendan McCollam, Stephen Rosen, and Ian Foster. 2016. Globus Auth: A research identity and access management platform. In *2016 IEEE 12th International Conference on e-Science (e-Science)*. IEEE, 203–212.

[60] Marie van de Sanden, Christine Staiger, Claudio Cacciari, Roberto Mucci, Carl Johan Hakansson, Adil Hasan, Stephane Coutin, Hannes Thiemann, Benedikt von St Vieth, and Jens Jensen. 2015. *D5.3: Final Report on EUDAT Services*. Technical Report. EUDAT.

[61] M Wan, R Moore, and A Rajasekar. 2009. Integration of cloud storage with data grids. In *3rd International Conference on the Virtual Computing Initiative*.

[62] Wasabi [n.d.]. Cloud Object Storage by Wasabi. https://wasabi.com Accessed June 1, 2020.

[63] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *7th Symposium on Operating Systems Design and Implementation*. ACM, New York, NY, 307–320.

[64] E James Whitehead and Meredith Wiggins. 1998. WebDAV: IETF standard for collaborative authoring on the Web. *IEEE Internet Computing* 2, 5 (1998), 34–40.

[65] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J.G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A C 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan Van Der Lei, Erik Van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* 3 (2016), 160018. https://doi.org/10.1038/sdata.2016.18

[66] Justin M Wozniak, Timothy G Armstrong, Michael Wilde, Daniel S Katz, Ewing Lusk, and Ian T Foster. 2013. Swift/T – large-scale applicatino composiion via distributed-memory dataflow processing. In *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 95–102.

[67] Esma Yildirim, Engin Arslan, Jangyoung Kim, and Tevfik Kosar. 2015. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *IEEE Transactions on Cloud Computing* 4, 1 (2015), 63–75.

GOVERNMENT LICENSE

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. 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. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.