Navigating the Unexpected Realities of Big Data Transfers in a Cloud-based World

Sergio Rivera
University of Kentucky
Lexington, Kentucky 40506
sergio@netlab.uky.edu

James Griffioen
University of Kentucky
Lexington, Kentucky 40506
griff@netlab.uky.edu

Zongming Fei
University of Kentucky
Lexington, Kentucky 40506
fei@netlab.uky.edu

Mami Hayashida
University of Kentucky
Lexington, Kentucky 40506
mhaya2@netlab.uky.edu

Pinyi Shi
University of Kentucky
Lexington, Kentucky 40506
pinyishi@netlab.uky.edu

Bhushan Chitre
University of Kentucky
Lexington, Kentucky 40506
bhushan@netlab.uky.edu

Jacob Chappell
University of Kentucky
Lexington, Kentucky 40506
jacob@netlab.uky.edu

Yongwook Song
University of Kentucky
Lexington, Kentucky 40506
ywsong2@netlab.uky.edu

Lowell Pike
University of Kentucky
Lexington, Kentucky 40506
pike@netlab.uky.edu

Charles Carpenter
University of Kentucky
Lexington, Kentucky 40506
charles@netlab.uky.edu

Hussamuddin Nasir
University of Kentucky
Lexington, Kentucky 40506
nasir@netlab.uky.edu

ABSTRACT

The emergence of big data has created new challenges for researchers transmitting big data sets across campus networks to local (HPC) cloud resources, or over wide area networks to public cloud services. Unlike conventional HPC systems where the network is carefully architected (*e.g.*, a high speed local interconnect, or a wide area connection between Data Transfer Nodes), today's big data communication often occurs over shared network infrastructures with many external and uncontrolled factors influencing performance.

This paper describes our efforts to understand and characterize the performance of various big data transfer tools such as `rclone`, `cyberduck`, and other provider-specific CLI tools when moving data to/from public and private cloud resources. We analyze the various parameter settings available on each of these tools and their impact on performance. Our experimental results give insights into the performance of cloud providers and transfer tools, and provide guidance for parameter settings when using cloud transfer tools. We also explore performance when coming from HPC DTN nodes as well as researcher machines located deep in the campus network, and show that emerging SDN approaches such as the VIP Lanes system can deliver excellent performance even from researchers' machines.

CCS CONCEPTS

• **Networks** → **Network management**; *Programmable networks*;

KEYWORDS

Data Transfer Tools, Software-Defined Networks, Big Data Flows

ACM Reference format:
Sergio Rivera, James Griffioen, Zongming Fei, Mami Hayashida, Pinyi Shi, Bhushan Chitre, Jacob Chappell, Yongwook Song, Lowell Pike, Charles Carpenter, and Hussamuddin Nasir. 2018. Navigating the Unexpected Realities of Big Data Transfers in a Cloud-based World. In *Proceedings of Practice and Experience in Advanced Research Computing, Pittsburgh, PA, USA, July 22–26, 2018 (PEARC '18),* 8 pages.
https://doi.org/10.1145/3219104.3229276

1 INTRODUCTION

In this new era of big data and cloud computing, the need to transmit data quickly and simply has become an increasingly important part of the workflow that researchers use. Big data has become foundational to virtually all areas of research. Even research areas that have historically not required significant computation or storage now find themselves dealing with massive data sets that need to be processed, analyzed, and shared. As a result, moving large data sets to/from the cloud has become both common-place for researchers and a major bottleneck that limits their research productivity.

Despite the increasing use of cloud and big data, researchers often struggle to make effective use of the available data transmission tools. This is due in large part to the complexity of data

PEARC '18, July 22–26, 2018, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-6446-1/18/07…$15.00
https://doi.org/10.1145/3219104.3229276

transmission. Achieving high speed data transmission depends on a wide range of factors such as the application used to transmit the data, the parameter settings of the application, the (cloud) provider where the data needs to be moved to/from, and local network characteristics, to name a few. Moreover, scientists are often unaware of the fact that different providers may apply aggressive rate limiting policies to prevent denial of service attacks and guarantee service to their ever growing customer base. The combination of all these factors easily becomes overwhelming and oftentimes results in the researcher opting for built-in "safe and good enough" default parameters which are often far from optimal for big data transmissions. For instance, rather than using a dedicated configurable command-line tool for their upload transfer to a cloud storage, scientists may be lured by the familiar easy-to-use web interfaces and decide to share their research data sets in the same way they upload personal files like pictures or documents through the web browser, ignoring factors like parallel transfers, file size, MTUs, dedicated network infrastructure, and types of hard-drives that could substantially improve/degrade the transmission performance of their science flows.

To alleviate this "gap", providers have developed feature rich REST APIs and Command Line Interface (CLI) tools that software developers can leverage to develop provider-specific or provider-agnostic client-side tools and plugins that can take advantage of the capabilities of the machine where the application is running such as number of cores, RAM, disk space, etc. Unfortunately, in most of the cases these tools pass the responsibility to the researcher who must figure out via a trial-and-error approach which parameters work best for his/her workflow.

In this paper, we take a look at the performance of various client-side tools designed for big data transfers. We consider (cloud) provider-agnostic tools as well as provider-specific tools and analyze both upload and download performance between the University of Kentucky (UK) Data Transfer Node (DTN) and various (cloud) storage providers, namely, AWS, Microsoft Azure Blob Storage, Dropbox, Google Drive, and UK's Ceph object store. We also explore the effects of various parameter settings on performance when downloading and uploading from/to desktop machines located deep inside our campus network using middlebox-free paths set up using the *VIP Lanes* [7] system (a programmable SDN-enabled network infrastructure that allows pre-authorized, very important packets (flows) to bypass rate-limiting firewalls, network address translators (NATs), and other types of middleboxes). Our evaluation considers performance and usability criteria such as tool reliability, the ways in which tool features affect throughput (*e.g.*, parallelism and chunked uploads), the way implementation details affect performance (*e.g.*, back-off mechanisms), or the way cloud providers' policies affect performance (*e.g.*, rate limits).

We present experimental results from our testing and evaluation that give insights into, and guidance regarding, the selection of tools and tool parameters that are best for transferring large datasets to/from different cloud providers.

The Rclone Tool: `rclone` offers the best overall performance and reliability when transferring data to/from any of the tested cloud providers, in large part due to its use of parallel transfers and an effective back-off mechanism. However, achieving high performance with the `rclone` tool requires careful tuning of its many parameters. Without careful tuning performance can quickly degrade.

Provider-specific CLI Tools: these tools focus on seamless and easy access to all of the services offered by the provider, rather than focusing on optimizing transfer speeds.

Fastest Providers: AWS and Google Drive performed at roughly twice the speed of other cloud providers.

Private/Local Storage Providers: UK's Ceph object store was faster than public cloud providers, but only by (roughly) a factor of two and could be slower depending on the tool and parameter settings used.

Upload vs. Download: Most providers' performance depended on the direction of transfer. Download speeds were faster than upload speeds for Dropbox and Google Drive. Somewhat surprisingly, uploads were noticeably faster than downloads for Azure. Upload and download speeds were roughly equal in the case of AWS.

Parallel Connections: Our results show that the number of parallel connections has a bigger impact on performance than chunk size. Moreover, parallel connections are only effective if the machine has as many cores as parallel connections, implying that tools with support for parallel connections have far less of an advantage over non-parallel tools when run on conventional desktop computers.

The paper is organized as follows. In Section 2 we introduce and describe the set of tools we used to move data to storage systems. Section 3 describes details of the experiments we ran to analyze data transfers as well as behaviors we observed while moving data to five storage systems. Section 4 explores the effects of various parameters of transfer applications on performance. We discuss related work on analysis of storage systems in Section 5, and lastly, Section 6 concludes the paper.

2 BIG DATA TRANSFER TOOLS

As storage systems continue to evolve in terms of services provided, architecture, design, and efficiency, more complex REST APIs are being made available to tool developers. As a result, a variety of data transfer tools that exploit these capabilities have emerged in the last couple of years, each with its own set of features including parallel transfers, chunked uploads, rate-limiting handlers, and storage systems supported. In this paper, we analyzed 6 of existing data transfer tools. A brief summary of each tool's characteristics is shown in Table 1.

Among the applications we evaluated we can roughly classify them into two main groups, namely, *provider-agnostic* and *provider-specific* data transfer tools. *Provider-agnostic* tools include `rclone` and `cyberduck` and offer the ability to work with multiple cloud providers through a single tool. These tools translate user requests into the appropriate API calls for the provider's storage system. *Provider-specific* tools, on the other hand, only work with one provider's storage system. Most provider-specific tools (with the exception of `gdrive-cli`) are developed by the corresponding storage provider. Rather than providing a generic one-size-fits-all application interface, these tools are developed to maximize access to the capabilities and features of one particular service provider.

Table 1: Features of data transfer tools analyzed

Tool	Storage Systems Supported	GUI	Prog. Language	Parallel Transfers	Chunked Uploads
rclone	Multiple (21)	Y	Go 1.6+	Y (param)	Y (param)
cyberduck	Multiple (16)	Y	Java 1.8	Y (param)	N
aws-cli	S3 protocol	N	Python 2.6+	Y (auto)	Y (auto)
gdrive-cli	Google Drive	N	Go 1.5+	N	Y (param)
azure-cli	Azure	N	Python 2.7+	Y (param)	Y (auto)
dbox-cli	Dropbox	N	Go	N	Y (16 MB)

2.1 Rclone

Rclone [10] is a provider-agnostic tool used to copy files and directories to and from more than 20 different storage systems and protocols. A GUI version was recently introduced. Rclone requires Go 1.6+ and supports a wide range of operations such as copy, move, delete, purge, list, duplicate removal, etc. Additionally, it has specific parameters per storage system that may have an impact on the throughput of data transfers. rclone allows multiple parallel file transfers up to the number of cores in the machine where it is running. However, by default, rclone sets this value to 4 parallel file transfers which is optimal for workstations but inefficient for high-end machines such as DTNs. For some cloud storage systems like Google Drive or Dropbox, rclone lets the user specify a chunk size value *CHS* which causes files larger than *CHS* to be divided into multiple chunks, where each chunk is in turn loaded in memory to increase upload speeds.

2.2 Cyberduck

Cyberduck [8] is another provider-agnostic tool. It is written in Java 1.8, and, until recently, was the only tool that provided both a browser-like GUI and a CLI version of their software. The set of operations supported is smaller than rclone's, but includes the basic ones for data transfers (*e.g.* upload, download, list, delete). Similar to rclone, cyberduck lets the user specify the number of parallel connections to use for transfers although in our experiments cyberduck only established a maximum of 10. Unlike rclone, it does not offer the possibility to split files into custom-sized chunks and buffer them in memory for performance improvement.

2.3 Gdrive-cli

To the best of our knowledge, Google has not released a CLI application to interact with Google Drive. Instead, initiatives from individuals have resulted in community driven projects trying to address the need for a provider-specific CLI for Google Drive. One of the more well-known community CLI-based tools for Google Drive is called gdrive-cli [13]. Even though the gdrive-cli tool is no longer maintained, the tool was one of the first ones to interact with Google Drive and provides compiled binaries for multiple platforms. Unlike the tools described so far, gdrive-cli does not support parallel transfers.

2.4 AWS-cli

AWS-cli [2] is a simple (yet powerful) python tool developed by Amazon to manage all 100+ services offered by AWS (*e.g.*, EC2, S3, Lambda, Glacier, etc). This CLI tool has a compact set of commands resembling UNIX syntax to manipulate the local file system with remote AWS buckets (where the data is pushed to). Even though the user cannot specify the number of parallel connections and chunk size, aws-cli supports multipart uploads per file, and, according to our observations, it generates up to 10 parallel connections. This CLI can be used with any storage system that supports the S3 protocol such as Ceph or DigitalOcean Spaces.

2.5 Azure-cli

Azure-cli [9] is a tool written in Python by Microsoft to manage all services provided by Azure, including its blob storage. As with all the previous tools, it supports basic operations on files and directories. Unlike aws-cli and gdrive-cli, it does allow the user to specify the maximum number of parallel transfers that may be created for each transfer.

2.6 DBX-cli

Written in Go, dbx-cli [6] has been the official command line tool for Dropbox users and team admins since 2016. It supports basic operations on individual files (not directories) via Dropbox's "Files" API. The tool is still in the early stage of development compared to other provider-specific applications, and currently cannot be adjusted via parameters.

3 EVALUATING TOOLS AND PROVIDERS

To understand the performance characteristics of the previously described tools, we performed various tests, transferring data between UK's Data Transfer Node (DTN) (that is directly connected the Internet 2 backbone at 40 Gbps) and five different storage providers; namely, Dropbox, AWS, Google Drive, Microsoft Azure Blob Storage, and UK's Ceph object store.

The following describes the experimental results we obtained by running tests from the UK DTN node to various cloud providers using one of the previously described tools.

3.1 Experiment Setup

The dataset we used consisted of 20 files with random contents (generated with the Linux tool dd and /dev/urandom random generator), each file being 200 MB in size, for a total of 4 GB of data. Each batch of uploads and downloads was performed 10 times, one after another. We computed the perceived throughput by dividing the dataset size over the number of seconds reported by the UNIX time command once the transfer was completed. Lastly, if the tool supported it, we set the chunk size for uploads to 16 MB, and the number of parallel connections to 20 (1 per file) in hopes of optimizing performance (see Section 4 for evaluation of other parameter settings).

3.2 Results

In the following we evaluate the reliability of the various tools (*i.e.*, did they work?), the performance improvements made possible by parallel data transfers, the upload and download speeds of the tools, and the benefits of using provider-specific tools. Our findings and recommendations are presented below based on the information presented in Figure 1.

Tool Reliability: Because many of these tools are relatively new and are evolving rapidly they tend to be rather fragile. Consequently, one cannot assume they will actually work reliably in all situations. For our tests, we say a tool "worked" if it successfully transferred the data – even if the transfer rate was very slow, or the tool (internally) retried the transfer multiple times. We define *reliability* as the ratio of the number of successful transfers to the number of attempted transfers. Two of the tools had very poor reliability. Specifically, `cyberduck` had several incomplete (*i.e.*, failed) upload transfers to Dropbox and Google Drive (8 out of 10 transfers failed), and `gdrive-cli` failed once per direction. In most cases, the failures appear to have been caused by an inability to effectively react to the provider's rate limiting policies. `cyberduck` never succeeded in transferring data to/from Azure because it was not able to authenticate and initiate the transfer. All other possible combinations of tools and storage systems worked without problems (*i.e.*, had 100% reliability). Part of their success is due to the fact that they effectively implemented back-off mechanisms to deal with the provider's rate limiting policies. For instance, using the `-vv` option in `rclone` to print debug messages, one can see that `rclone` implements an exponential back-off approach that works well and results in complete transfers albeit at the expense of some extra delay. The `rclone` back-off mechanism increases the wait times for a new retransmission every time a particular file or chunk is rejected by the storage system. Other back-off mechanisms, such as the one used in `cyberduck` are far less effective. For example, the `cyberduck` back-off implementation is based on simple retries; the user passes a "number of retries" parameter that `cyberduck` is intended to honor whenever a file is rejected. However, from our observations the maximum number of retries was much lower than values passed in which explains why `cyberduck` transfers were incomplete in multiple cases.

Parallel Transfers: Parallel transfers, as one might expect, can improve performance significantly. However, our experimental results show that the effectiveness of parallel transfers is directly linked to the number of cores on the client machine (see Section 4). This helped our UK DTN node achieve some of the best performance numbers. With 32 cores and a dataset that consists of only 20 files, it was able to easily handle the 20 parallel transfers needed to maximize throughput. Of all the tested tools, `dbx-cli` was the only one that used a single connection to upload or download all the files. In the upload case, since our files were larger than 16 MB, `dbx-cli` divided each file into 16 MB chunks, and pushed each of them one after another. As a result, it became notoriously slower when compared to other tools that supported parallel transfers. In fact, as shown in Figure 1a, the average upload rate for `dbx-cli` never exceeded 50 Mbps with a very small standard deviation. This is exceptionally low throughput considering that the DTN node (with 40 Gbps interfaces) was the source of the file transfer. Nevertheless, even with parallel transfers, `cyberduck` and `rclone` were not able to achieve fast transfers to Dropbox– when compared to other providers – and could never take advantage of the high-end hardware of the DTN.

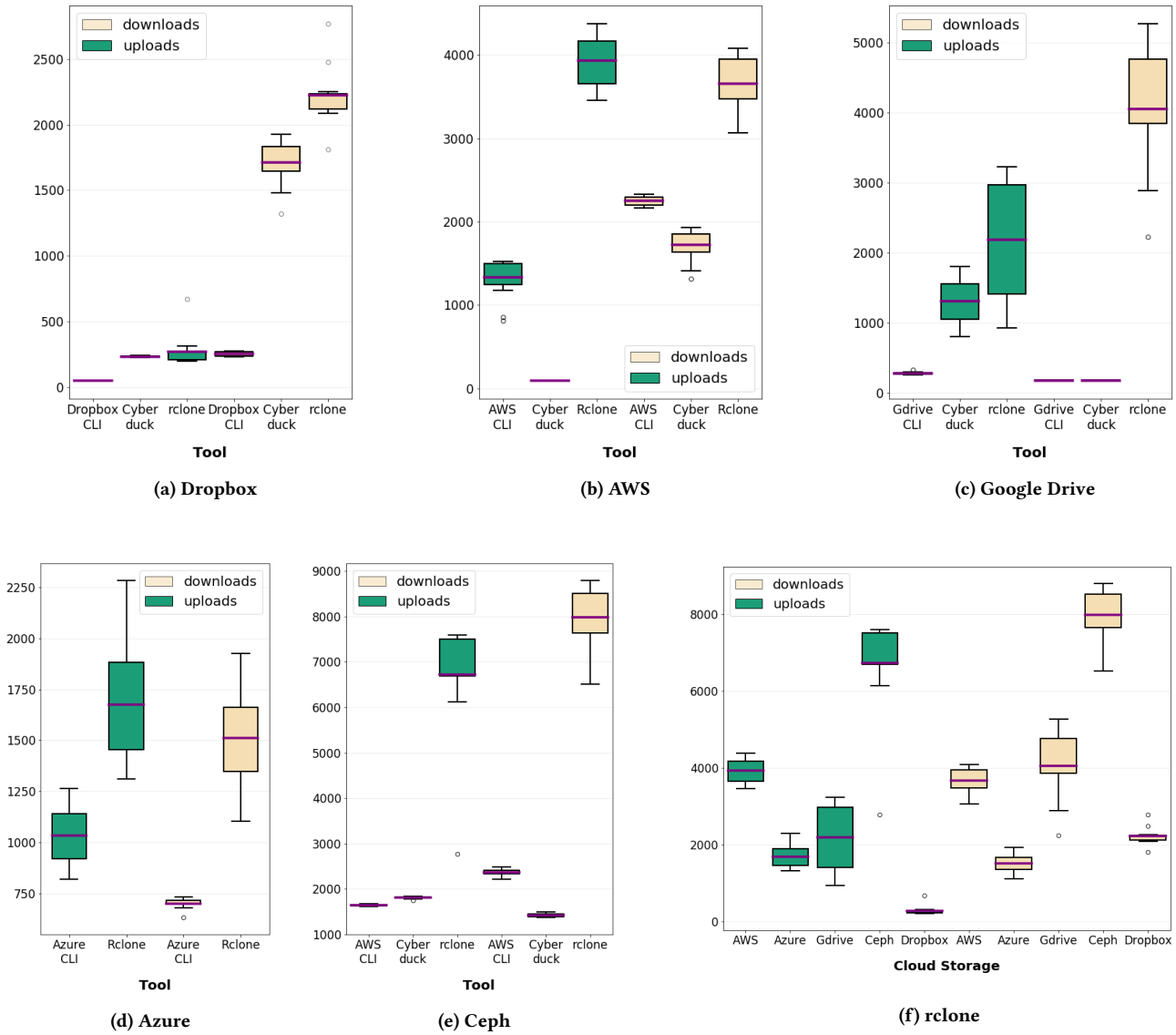
Provider-specific Tools vs. Provider-agnostic Tools: Even though one might think that provider-specific tools are optimized for their associated cloud provider – *e.g.*, `aws-cli` would outperform all other tools for transfers with AWS– we found this is not the case. Provider-specific tools are just a convenient interface for management of *all* the services supported from a particular cloud provider (*e.g.*, virtual machine reservation, billing, account management, etc). They tend not to be optimized for file transfers. In fact, for all provider-specific tools, the set of data transfer commands supported makes it easy to invoke operations at the provider, as compared to other tools like `rclone`. However, `rclone` outperforms all `cli` tools for both uploads and downloads (see Figure 2), making it the best and most reliable tool, albeit, slightly harder to use.

Upload vs. Download: Providers' performance often depends on the direction of transfer. Roughly speaking, download speeds were faster than upload speeds for Dropbox and Google Drive, uploads were noticeably faster than downloads for Azure, and upload and download speeds were roughly equal in the case of AWS. The following takes a closer look at the upload/download performance:

Dropbox: We observed (Figure 1a) that downloads were significantly faster than uploads (mean download 7-8x faster using `cyberduck`, 9x in `rclone`, and 5x in `dbx-cli`). The poor performance when uploading from the DTN (below 500 Mbps in 95% of the cases) is caused by the fact that Dropbox does not allow upload of multiple chunks simultaneously. Additionally, even when chunks reach the provider's end-point, Dropbox may reject the file transfer due to multiple files competing for a lock in order to be verified. From our experiments, we noticed that Dropbox forces retransmissions whenever the tool uses parallel transfers (*i.e.*, when using `cyberduck` and `rclone`). However, if no retransmission occurs, one can obtain the best performance which in our case was 668.643 Mbps (outlier dot in `rclone`) – still the lowest upload throughput across all storage systems.

AWS: Our experiments show that throughput relation between downloads and uploads varies depending on the tool used (Figure 1b). When using `aws-cli`, downloads were always 1.5 times faster than uploads, measuring above the 2 Gbps and 1 Gbps marks, respectively; secondly, `cyberduck` showed a vast difference, being 1 order of magnitude faster when downloading. However, `cyberduck` was the tool that provided the worst performance in both types of transfers. Lastly, `rclone` was the only tool that reported better mean throughput for uploads than downloads. Nevertheless, both directions were close to the 4 Gbps mark with a relatively small difference (~0.3 Gbps) between their mean throughput. For this storage system, `rclone` significantly outperformed `cyberduck` and `aws-cli` and should be the tool of choice when moving data – the higher number of parallel transfers (20) significantly impacted the final throughput compared to the maximum of 10 parallel transfers used by the other tools.

Google Drive: As seen in Figure 1c, only `rclone` reported throughput faster for downloads than uploads. Notably, download speeds using `rclone` (~4 Gbps) were more than 20 times faster than `gdrive-cli` and `cyberduck` (both reporting mean speeds around 177 Mbps). The latter is surprising given that `cyberduck` allows up to 10 parallel transfers whereas `gdrive-cli` does not. In terms of uploads, `gdrive-cli` experienced equal throughput as downloads, whereas `cyberduck` was around 8 times faster than its mean download performance.



(a) Dropbox (b) AWS (c) Google Drive

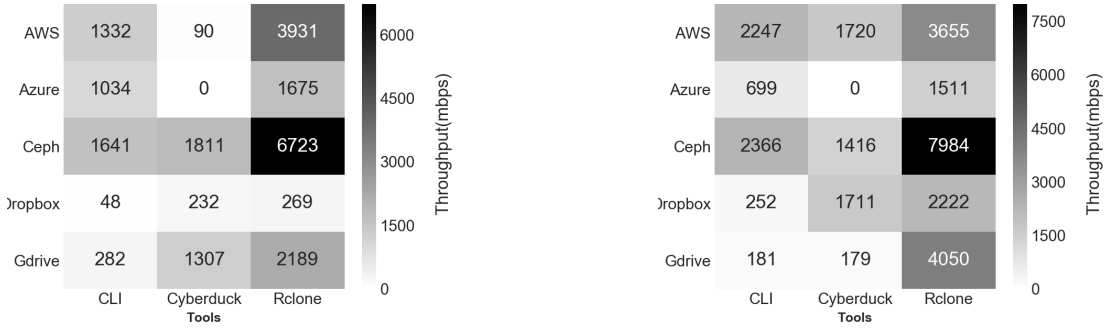
(d) Azure (e) Ceph (f) rclone

Figure 1: Throughput measurements (in Mbps) to five different storage systems

Azure: Figure 1d shows that uploads perform better than downloads for both `rclone` and `azure-cli`. Downloads were particularly slow using `azure-cli` with speeds sitting below 750 Mbps and a low standard deviation of 38.75 Mbps. The situation improved when we ran the experiments using `rclone`, almost doubling `azure-cli`'s performance with mean speeds around 1.5 Gbps. On the other hand, for uploads, `azure-cli` reported better mean speeds above 1 Gbps, although it was again outperformed by `rclone`, whose upload throughput ranged between ~1.3 Gbps in the worst case and ~2.2 Gbps in the best case.

UK's Ceph object store: Our local object store got the best performance when using `rclone` for pushing and pulling data with a mean throughput of ~6.8 Gbps and ~8.0 Gbps, respectively (see Figure 1e). It is important to note that in one of the experiments, `rclone` had to retransmit part of the data set causing the performance to significantly drop to ~2.7 Gbps. Still, `rclone` provided yet again the best throughput in both types of transfer when compared to `aws-cli` and `cyberduck`, which reported transfer rates between 1.3 Gbps and 2.4 Gbps. As in other cloud storage systems, the parallel transfer mechanism of `rclone` provided a significant boost to the final throughput when moving datasets and we conclude it is the tool that should be used when interacting with local Ceph storage systems.

Service Providers. Unsurprisingly, the best throughput was obtained when transferring data to/from UK's Ceph object store. As



(a) Uploads **(b) Downloads**

Figure 2: Mean throughput heatmaps for data transfer tools to different storage systems

mentioned earlier, `rclone`'s parallel transfer mechanism allowed us to reach speeds approaching 9.0 Gbps, the best performance we record across all five storage systems(see Figure 1f). With respect to public cloud solutions, our results indicate that transfers originated from our DTN to AWS and Google Drive are generally faster than Dropbox and Azure using `rclone`. Therefore, we will analyze them pair-wise. In Google Drive and AWS, our observations indicated that the upload and download standard deviation for the former (845.20 Mbps and 938.93 Mbps, respectively) was significantly bigger than for the latter (323.19 Mbps and 336.32 Mbps, respectively). Consequently, we expect higher variations in throughput when using Google Drive and a more stable final speed when transferring to/from AWS. When comparing Dropbox and Azure storage systems, we noticed that Dropbox was outperformed by Azure for uploads. For downloads, however, `cyberduck` with Dropbox achieved roughly the same speed as `rclone` with Azure while `rclone` with Dropbox obtained the best performance across this pair of storage systems.

4 EVALUATING END SYSTEM LOCATION

In this section, we explore performance as it relates to the location of the researcher's machine in the campus network and the tool parameters used in the transfer. In the previous section we focused on transfers from the HPC Data Transfer Node (DTN) at the edge of the campus network. In this section, we explore performance of two types of *end system nodes* where researchers "live and work": (file) *server nodes* that have substantial computing power, and *desktop nodes* that are more resource constrained. In both cases, we assume that the nodes are located deep inside the campus network. One major problem of transferring data from these machines to the cloud is that the path to the cloud has to go through various middleboxes on campus that provide necessary security functions (say via deep packet inspection), and at the same time, pose a serious bottleneck to cloud data transfers. To address this problem, the University of Kentucky VIP Lanes system [7] takes advantage of the SDN switches/routers in the campus network to set up middlebox-free paths directly to the campus edge router for pre-authorized and trusted users. Our evaluations use these SDN-paths to provide high-speed bottleneck-free connections to cloud storages from machines located deep inside the campus network.

From the previous section, we saw that `rclone` outperforms all other cloud transfer applications. Consequently we have recommended `rclone` to researchers on campus as the tool of choice. Among cloud storage providers, Google Drive has become very popular among researchers both because of its good performance and cost (free unlimited storage). Consequently, our analysis will focus on using `rclone` to transfer data from campus file servers and desktops to Google Drive. By pairing `rclone` with an SDN-enabled VIP Lanes campus network and tuning the `rclone` parameters (*e.g.* number of parallel transfers, chunk size), we have been able to obtain significantly better throughput from end system nodes, sometimes comparable to the performance from the DTN, even from dedicated server nodes and desktop nodes deep in the campus network.

4.1 Experiment Setup

For this set of experiments we added two more source nodes shown in Figure 3 as *Aztec Desktop* and *Flint Server*. The characteristics of the nodes are as follows:

- **Flint Server:** A server machine with a high-speed path (and fewer hops) to the Internet. It has a powerful processor (an Intel(R) Xeon(R) CPU E5-2650L v3) with 48 cores running at 1.80GHz, 180 GB of RAM, and a 10 Gbps network interface with jumbo frames (*i.e.* MTU 9000) enabled.
- **Aztec Desktop:** A desktop workstation in our laboratory running Ubuntu 16.04. This node has an Intel(R) Core(TM) i5-4570S processor with 4 cores running at 2.90GHz, 8 GB of RAM, a 1 Gbps network interface with jumbo frames enabled.

Instead of using one data set with a fixed number of files as we did in the previous section, we created datasets of different file sizes and different number of files. Specifically, we generated three datasets, each consisting of one single file varying in size: 1GB, 10GB, and 100GB. We then divided each of these files into 10 and 50 equally-sized files using the `split` command line utility to obtain six additional datasets, for a total of nine data sets.

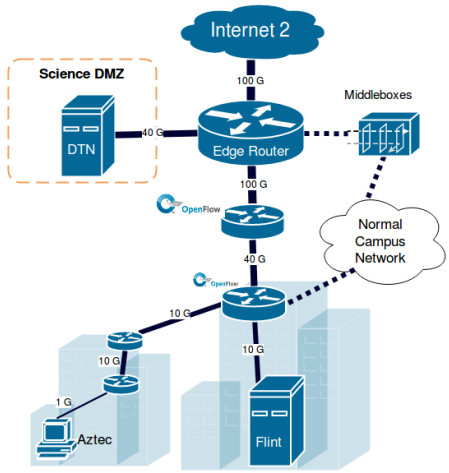


Figure 3: Location of source nodes in campus network

We uploaded each of the data sets from all three locations (*i.e.* Aztec Desktop, Flint Server, and DTN) to Google Drive 4 times and downloaded from Google Drive to these locations also 4 times. We recorded the throughput once the transmission finished. For every push and pull operation, we changed the numbers of parallel connections (4, 8, 16 or 32) and chunk size parameters (4 MB, 8 MB, 16 MB and 32MB). Due to the 750GB upload limit per account imposed by Google Drive, we limited the tests for the 100 GB data sets to only 16 and 32 transfers.

4.2 Results

Viplanes Boost: By enabling high-speed paths for big data science flows, servers can, at least in some cases, achieve speeds close to their maximum capacity and often similar to speeds obtained on the high-end DTN node which are sufficient to move big data to a cloud storage system very quickly. For instance, as shown in Table 2, some of our measurements recorded speeds greater than 700 Mbps from the Aztec Desktop machine, which is approaching the theoretical maximum of 1 Gbps, and is about 5-7x faster than going through the normal campus network (~100-150 Mbps), and orders of magnitude faster than the speeds recorded by others [14] (~600 KB/s in the best case) when moving data to other cloud storage systems from a campus machine.

Table 2: Upload and download speeds by location (Mbps)

Location-Dir	Mean	Std Dev	Maximum
Aztec Desktop-up	395	159	734
DTN-up	854	903	5664
Flint Server-up	437	381	2164
Aztec Desktop-down	385	176	768
DTN-down	1839	1226	5204
Flint Server-down	1420	799	3986

Chunk Size: When uploading large files, it is often useful to chunk the file into multiple smaller pieces as retransmissions incur less overhead. Instead of retransmitting the whole file, only a small piece is retransmitted. `rclone` allows the user to specify the size of each generated chunk to be loaded in memory by the thread in charge of transmitting the file. Figure 4 shows six summary statistics (min, first quartile, median, mean, third quartile, and max) for throughput (in Mbps) for 4 different chunk sizes for the DTN, Flint Server, and Aztec Desktop. Based on the figure, we can observe that the mean values increase with chunk size. Increasing the chunk size, however, had less impact on throughput than increasing the number of parallel transfers. For instance, for the DTN, when the the chunk size changes from 4, to 8, to 16 and then to 32 MB, the mean throughput changes from 556, to 727, to 1046 and then to 1062 Mbps, respectively, whereas when the number of parallel transfers changes from 4, to 8, to 16 and then to 32, the mean throughput changes from 425, to 651, to 1077, and then to 1187 Mbps, respectively.

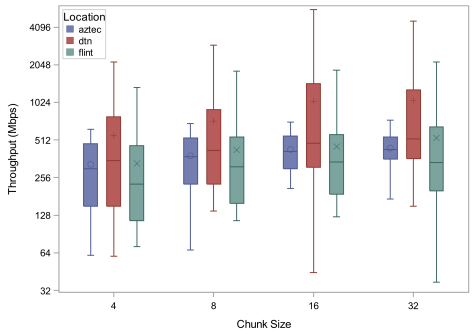


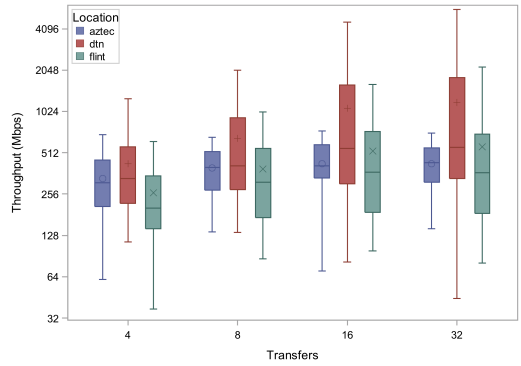
Figure 4: Upload speed by chunk size (log scale)

Parallel Connections and Number of Cores: As we pointed out in Section 3, tools that were able to create multiple parallel connections yielded better performance in all storage systems. The connections parameter is particularly important if one wants to take advantage of the core count found in high-end machines (*e.g.*, the DTN). As seen in Figure 5, using a lower number of threads than the number of cores produces slower speeds for the capabilities of the source node. The influence of this parameter is more noticeable at the DTN and Flint Server for both uploads and downloads. For instance, the maximum throughput from the Flint Server while pushing data to Google Drive measured 617 Mbps, 1013 Mbps, 1608 Mbps, and 2164 Mbps when increasing the number of parallel transfers from 4 to 8 to 16 to 32 respectively. A similar behavior can be seen while analyzing the DTN as both nodes have >= 32 cores.

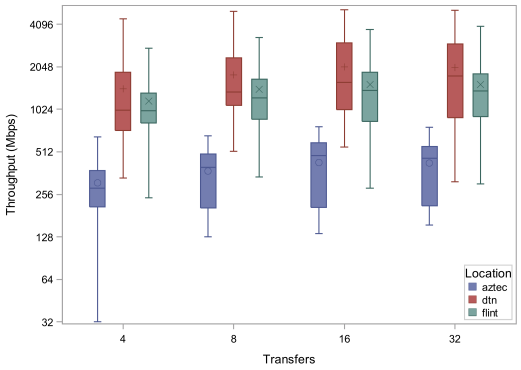
5 RELATED WORK

Analyzing the network performance of cloud storage systems has become more relevant in recent years given that many companies and research institutions are outsourcing their storage needs and trying to understand which provider would offer the best quality of service for their business/research operations.

Persico *et al.* [12] describes performance obtained while downloading files from AWS S3, considering factors like customer location, cloud datacenter region, type of storage within AWS and file sizes. Shen *et al.* [14] evaluated the REST API features of four storage systems (Baidu, Kingsoft, BOX and Dropbox). The authors



(a) Upload



(b) Download

Figure 5: Throughput by number of connections (log scale)

did not explore however, the impact of factors like chunking and parallel connections on the final throughput.

Researchers at Politecnico di Torino also present a set of benchmarks for different cloud storage systems [4, 5]. They focused on characteristics associated with the synchronization of local and remote files/folders (*e.g.*, how often storage systems check for a change in local file/directory that needs to be reflected in the remote side, bundling and encryption techniques) using native (GUI) clients running on Windows.

None of these studies explored performance from HPC DTN systems or high end researcher nodes located inside the campus network with the need to transfer large research data sets. Instead, they focused on personal workloads. Moreover, they did not utilize high-performance campus network channels and thus experience network bottlenecks that are now possible to avoid with SDN networks. Our work shows that with SDN enabled systems like VIP Lanes, even end systems located deep in the campus network can achieve cloud transfer rates approaching the maximum possible and similar to HPC DTN nodes.

Researchers at the University of Utah analyzed the number of connections and chunk size for `rclone` uploads to Google Drive in [11]. Our findings aligned with theirs, but go beyond to show the performance to other cloud providers, as well as exploring the performance of other transfer tools, and an expanded parameter space. Finally, various papers [1, 3] have described the performance of Globus' file transfer mechanism, which supports premium storage connectors to certain cloud provides for a fee. While the performance numbers are quite impressive, it is primarily designed for DTN-to-DTN transfers, requires connecting to globus, has limited support for cloud providers, and is cost prohibitive for some institutions.

6 CONCLUSION

We have witnessed that more and more cloud storage systems have become available and correspondingly, more transfer tools can be used to upload data to and download data from the cloud. It becomes a challenging problem to select which cloud storage system and which tool to use. In this paper, we performed a comprehensive study of performance of several cloud storage systems and data transfer tools. Our study focused on performance of transferring big data set and used the DTN nodes as the source node to prevent the influence of middleboxes on campus networks. We also explored the parameter spaces when using `rclone` for data transfer between Google Drive and desktops located deep inside campus network. This work gives insights into and provides guidance about choosing an appropriate application and setting best parameters when transferring large datasets from/to the clouds.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grants ACI-1541380, ACI-1541426, and ACI-1642134. We thank UK CCS/ITS-RC for letting us use their facilities for our tests and Vira Pravosud for her assistance with graph generation.

REFERENCES

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. 2005. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*.

[2] Amazon. 2018. AWS Command Line Interface. https://aws.amazon.com/cli/. (2018).

[3] J. Basney and P. Duda. 2007. Clustering the Reliable File Transfer Service. In *Proceedings of the 2007 TeraGrid Conference*.

[4] E. Bocchi, I. Drago, and M. Mellia. 2017. Personal Cloud Storage Benchmarks and Comparison. *IEEE Transactions on Cloud Computing* 5, 4 (Oct 2017), 751–764. https://doi.org/10.1109/TCC.2015.2427191

[5] E. Bocchi, M. Mellia, and S. Sarni. 2014. Cloud storage service benchmarking: Methodologies and experimentations. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. 395–400. https://doi.org/10.1109/CloudNet.2014.6969027

[6] Dropbox. 2018. dbxcli: A command line tool for Dropbox users and team admins. https://github.com/dropbox/dbxcli. (2018).

[7] J. Griffioen, K. Calvert, Z. Fei, S. Rivera, J. Chappell, M. Hayashida, C. Carpenter, Y. Song, and H. Nasir. 2017. VIP Lanes: High-Speed Custom Communication Paths for Authorized Flows. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–9. https://doi.org/10.1109/ICCCN.2017.8038429

[8] D. Kocher, Y. Langisch, and J. Malek. 2018. Cyberduck. https://cyberduck.io/. (2018).

[9] Microsoft. 2018. Azure CLI 2.0. https://docs.microsoft.com/en-us/cli/azure/?view=azure-cli-latest. (2018).

[10] Nick Craig Wood. 2018. Rclone - rsync for cloud storage. https://rclone.org/. (2018).

[11] The University of Utah. 2018. Exploring the Effects of Options on Performance. https://www.chpc.utah.edu/documentation/software/rclone.php. (2018).

[12] V. Persico, A. Montieri, and A. PescapÃĺ. 2016. On the Network Performance of Amazon S3 Cloud-Storage Service. In *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. 113–118. https://doi.org/10.1109/CloudNet.2016.16

[13] Petter Rasmussen. 2017. Google Drive CLI client. https://github.com/prasmussen/gdrive. (2017).

[14] P. Shen, K. Guo, and M. Xiao. 2014. Measuring the QoS of Personal Cloud Storage. In *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. 1–6. https://doi.org/10.1109/ICCCNT.2014.6963099