The science behind the report:

Upload files faster with IBM Aspera High-Speed File Transfer for IBM Cloud Object Storage

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Upload files faster with IBM Aspera High-Speed File Transfer for IBM Cloud Object Storage.

We concluded our hands-on testing on April 19, 2019. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on January 23, 2019 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

We ran each upload test three times and report the median results below. As our results show, IBM® Cloud Object Storage with IBM Aspera® High-Speed File Transfer was able to transfer the files up to 12 times faster than Amazon® S3 with Transfer Acceleration and up to 3.3 times faster than Microsoft Azure Blob storage with AzCopy.

IBM Cloud Object Storage vs. Amazon S3

20GB, 100ms latency test

Time to complete 20GB file transfer with increasing packet loss

	0% loss	0.1% loss	1% loss	2% loss
IBM Aspera High-Speed File Transfer (seconds)	28.39	30.78	29.97	29.77
Amazon S3 with Transfer Acceleration (seconds)	127.07	162.29	189.26	211.72
IBM transfer rate	**4.48x faster**	**5.27x faster**	**6.31x faster**	**7.11x faster**

20GB, 200ms latency test

Time to complete 20GB file transfer with increasing packet loss

	0% loss	0.1% loss	1% loss	2% loss
IBM Aspera High-Speed File Transfer (seconds)	32.5	31.29	33.89	31.67
Amazon S3 with Transfer Acceleration (seconds)	221.69	302.07	353.17	382.23
IBM transfer rate	**6.82x faster**	**9.65x faster**	**10.42x faster**	**12.07x faster**

50GB, 100ms latency test

Time to complete 50GB file transfer with increasing packet loss				
	0% loss	**0.1% loss**	**1% loss**	**2% loss**
IBM Aspera High-Speed File Transfer (seconds)	85.65	78.65	84.43	86.66
Amazon S3 with Transfer Acceleration (seconds)	319.28	368.63	410.68	453.42
IBM transfer rate	**3.73x faster**	**4.69x faster**	**4.86x faster**	**5.23x faster**

50GB, 200ms latency test

Time to complete 50GB file transfer with increasing packet loss.				
	0% loss	**0.1% loss**	**1% loss**	**2% loss**
IBM Aspera High-Speed File Transfer (seconds)	89.54	90.34	88.15	86.72
Amazon S3 with Transfer Acceleration (seconds)	561.1	688.57	832.26	947.34
IBM transfer rate	**6.27x faster**	**7.62x faster**	**9.44x faster**	**10.92x faster**

IBM Cloud Object Storage vs. Microsoft Azure Blob storage

20GB upload at 100ms latency

Time to complete 20GB file transfer with increasing packet loss				
	0% loss	**0.1% loss**	**1% loss**	**2% loss**
IBM Aspera High-Speed File Transfer (seconds)	28.39	30.78	29.97	29.77
Microsoft Azure Blob storage with AzCopy (seconds)	60.9	60.84	65.22	69.56
IBM transfer rate	**2.15x faster**	**1.98x faster**	**2.18x faster**	**2.34x faster**

20GB upload at 200ms latency

Time to complete 20GB file transfer with increasing packet loss				
	0% loss	**0.1% loss**	**1% loss**	**2% loss**
IBM Aspera High-Speed File Transfer (seconds)	32.5	31.29	33.89	31.67
Microsoft Azure Blob storage with AzCopy (seconds)	65.18	70.4	78.36	79.08
IBM transfer rate	**2.01x faster**	**2.25x faster**	**2.31x faster**	**2.50x faster**

50GB upload at 100ms latency

Time to complete 50GB file transfer with increasing packet loss.

	0% loss	0.1% loss	1% loss	2% loss
IBM Aspera High-Speed File Transfer (seconds)	85.65	78.65	84.43	86.66
Microsoft Azure Blob storage with AzCopy (seconds)	281.78	248.60	266.53	258.07
IBM transfer rate	**3.29x faster**	**3.16x faster**	**3.16x faster**	**2.98x faster**

50GB upload at 200ms latency

Time to complete 50GB file transfer with increasing packet loss.

	0% loss	0.1% loss	1% loss	2% loss
IBM Aspera High-Speed File Transfer (seconds)	89.54	90.34	88.15	86.72
Microsoft Azure Blob storage with AzCopy (seconds)	247.36	277.97	279.85	292.97
IBM transfer rate	**2.76x faster**	**3.08x faster**	**3.17x faster**	**3.38x faster**

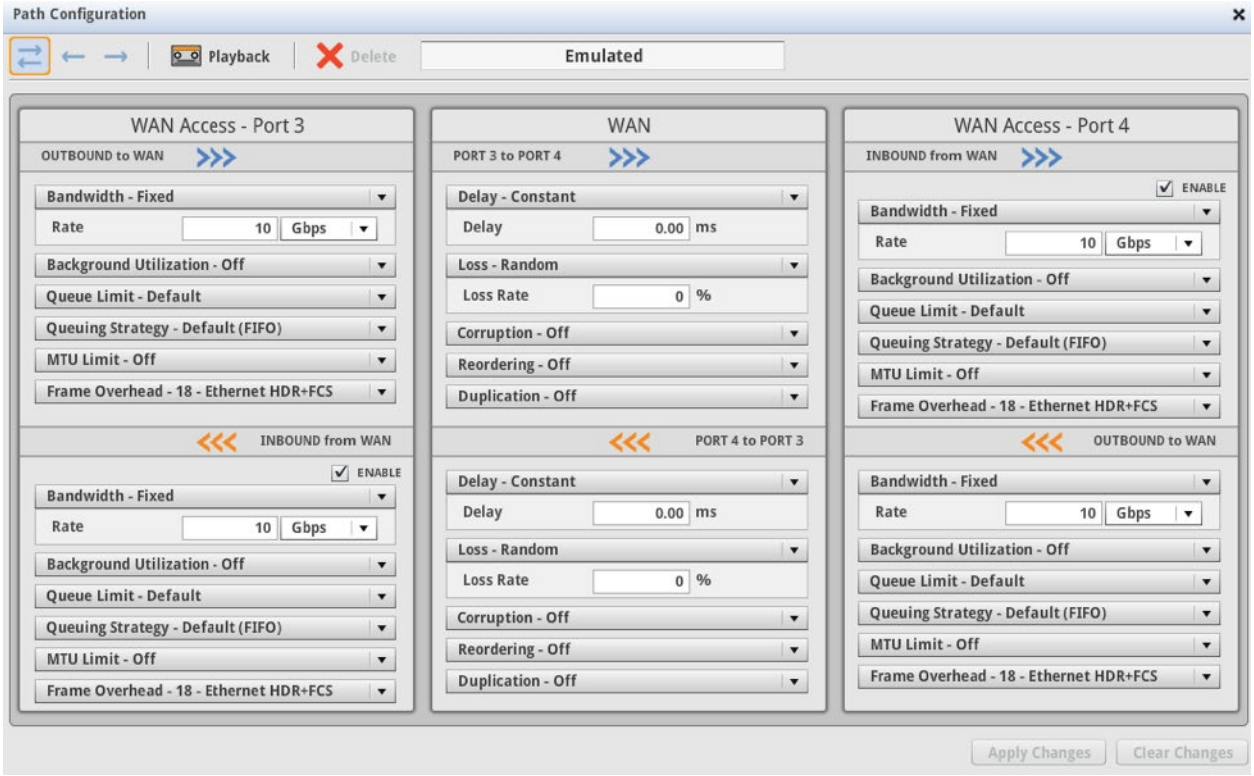
System configuration information

The table below presents detailed information on the systems we tested.

Server configuration information	Lenovo™ ThinkSystem™ SR530
General	
Number of physical processors	1
CPU cores/threads	12 / 24
CPU	
Vendor	Intel®
Name	Xeon® Gold 5118
Stepping	4
Core frequency (GHz)	2.30
Front-side bus frequency (MHz)	2400
L1 Cache	768KiB
L2 Cache	12MiB
Platform	
Vendor	Lenovo
Motherboard model number	7X08CTO1WW
Motherboard serial number	W7ZS87L00HB
BIOS revision	1.40
Firmware revision	1.90
Memory module(s)	
Vendor and model number	Samsung
Type	DIMM DDR4 Synchronous
Clock (MHz)	2400 (0.4ns)
Capabilities	ECC
Size	32GiB
Number of RAM modules	5
System RAM	96 GB
Hard disk	
Number of disks in system	1
Size	1.8TiB
Type	MegaRAID Tri-Mode SAS3508
Controller	Lewisburg SATA Controller [AHCI mode]

Server configuration information	Lenovo™ ThinkSystem™ SR530
Operating system	
Name	Ubuntu
Build number	18.04.1 LTS
File system	EXT4
Kernel	4.15.0-34-generic
Network card/subsystem	
Ethernet Interface 1	Intel 82599ES 10-Gigabit SFI/SFP+
Ethernet Interface 2	Intel X722 for 1GbE Copper

How we tested



Test environment

We conducted each of the 20GB and 50GB tests from a machine in an Aspera laboratory in Emeryville, CA. The machine ran Ubuntu 18.04.1 LTS. We instructed IBMa to physically route network traffic from the machine through a Netropy 10G2 WAN emulator appliance and out to the internet via the laboratory's 10Gb connection, enabling us to simulate long-distance network conditions.

We set up object storage buckets on IBM Cloud Object Storage, Amazon S3, and Microsoft Azure Blob storage in the US South (Texas), US East 2 (Ohio), and South Central US (Texas) regions respectively. In order to simulate 100ms and 200ms total round-trip latency in our tests, we first determined the baseline average total round-trip latency from the test environment to each of the target cloud storage endpoints as follows:

1. For IBM COS and AWS S3, we used the ping command to measure average round-trip latency to each respective storage endpoint:
   a. The average round-trip latency to IBM COS's US South regional endpoint (s3.us-south.objectstorage.softlayer.net) was approximately 46ms.
   b. The average round trip latency to the Amazon S3 US East 2 regional endpoint (s3.us-east-2.amazonaws.com) was approximately 60ms.
2. For Azure Blob storage, we used the nmap nping command to measure average round-trip latency. Because the Azure firewall does not respond to ICMI, we ran the following command to use TCP instead:

   ```
   nping -c 5 --tcp-connect -p 443 principledtech.blob.core.windows.net
   ```

   The average round trip latency to the Azure object storage endpoint was approximately 46ms.

After determining baseline latency values, we simulated the additional latency required for each file transfer to hit 100ms or 200ms.

We used the following Linux command to generate a 20GB binary file with random content to use for all test uploads.

```
dd if=/dev/urandom of=file20gb.bin count=1024 bs=20m
```

We used the same command to generate a 50GB binary file:

```
dd if=/dev/urandom of=file20gb.bin count=1024 bs=50m
```

Using the Netropy appliance

We used the following steps to change the simulated network conditions in the Netropy appliance.

1. Connect to the Netropy browser-based interface.
2. Ensure the correct emulation engine is selected, and set the emulation to On.
3. To open the Path Configuration screen shown above, click the "Emulated" graphic at the top of the screen.
4. Modify the delay and packet loss in the central panel as required for each test configuration:
   a. For delay, put half of the desired total simulated latency in each direction. For example, if adding 50ms total simulated latency, specify 25ms delay in each direction.
   b. For loss rate, put the full packet loss percentage in each direction. For example, if simulating 1% packet loss, both directions should be set to 1% random loss rate.
5. Click Apply Changes.

Running the 20GB tests on IBM Cloud Object Storage using the IBM Aspera SDK

Below are the steps we took to run tests using the IBM Aspera SDK. IBM provided us with a custom script for our testing; however, as of this writing, IBM has implemented modifications to their SDK to include the script's features, and does not recommend that end users run the script for anything other than replicating our testing. Instead, IBM recommends using the fully supported SDK from their website at https://developer.ibm.com/aspera/docs/ibm-aspera-faspstream-sdk-3-8-0/.

1. Use an SSH client to connect to the test machine.
2. To install the pip package manager, run the following command:

```
apt install python3-pip
```

3. To install the IBM Cloud Object Storage SDK for Python, run the following command:

```
pip3 install ibm-cos-sdk
```

4. To install the proprietary Aspera SDK add-on for the IBM Cloud Object Storage SDK for Python, run the following command:

```
pip3 install cos-aspera
```

5. Set the packet loss and latency parameters in the Netropy web interface as required for the test configuration.
6. Run the script from the appendix on page 9, and record the result in seconds.
7. Repeat steps 5 and 6 as needed to complete all tests. We ran each test configuration three times, and we reported the median result.

Running the 20GB tests on Amazon AWS using the S3 SDK with Transfer Acceleration

1. In the AWS Management Console, under Test, navigate to the S3 bucket.
2. Click Properties, and enable Transfer Acceleration.
3. Use an SSH client to connect to the test machine.
4. Run the following command to install the AWS S3 SDK for Python:

```
pip3 install boto3
```

5. Create a file called `config` with the following content:

```
[default]
region = us-east-2
s3 = use_accelerate_endpoint = true
```

6. Create a file called `credentials` with the following content, replacing the access keys with your own:

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

7. Move the files from step 5 and 6 to ~/.aws/
8. Set the packet loss and latency parameters in the Netropy web interface as required for the test configuration.
9. Edit the following script to include your target S3 bucket name. Run the script, and report the result in seconds.

```
import boto3, time, random, string
```

```
s3 = boto3.client('s3')
bucketname = '[YOUR BUCKET NAME]'
# generate random string of length 5 for destination object name in bucket
rand_string = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(5))

def upload_file(filename):
    s3 = boto3.client('s3')
    try:
            print ("Uploading file 20gb file")
            start = time.time()
            transfer_config = boto3.s3.transfer.TransferConfig()
            transfer = boto3.s3.transfer.S3Transfer(client=s3, config=transfer_config)
            transfer.upload_file(filename, bucketname, rand_string)
            end = time.time();
            print ("Transfer successful!")
            print(str(round(end-start, 2)) + 's')
    except Exception as e:
            print ("Error uploading %s" % (e))
upload_file('file20gb.bin');
```

10. Repeat steps 8 and 9 as needed to complete all tests. We ran each test configuration three times, and we reported the median result.

Running the 20GB tests on Azure Blob storage with AzCopy

1. Use an SSH client to connect to the test machine.
2. To install AzCopy 8 for Linux, run the following commands:

```
wget -O azcopy.tar.gz https://aka.ms/downloadazcopylinux64
tar -xf azcopy.tar.gz
sudo ./install.sh
```

3. Run the following command, replacing account name, bucket name, and access key with your own. Record the result in seconds:

```
time azcopy --source file20gb.bin --destination https://[ACCOUNT NAME].blob.core.windows.net/[BUCKET
NAME]/file20gb.bin --dest-key [YOUR ACCESS KEY]
```

4. Repeat step 3 as needed to complete all tests.

Running the 50GB tests

To complete the 50GB tests with each cloud provider, repeat the steps for the 20GB tests, replacing relevant information for the 20GB file with information for the 50GB file.

Appendix: IBM Aspera High-Speed File Transfer script

To test the IBM Cloud service with IBM Aspera High-Speed File Transfer, we used a script from IBM. We present the full text of the script below, though IBM does not recommend you use this script for your actual environment or applications. IBM offers an Aspera SDK they claim offers the same functionality as this script in a more user-friendly package. We have not tested the newest Aspera SDK, so we cannot say whether using it instead of the script we provide below will affect your results. To learn more about the SDK, visit https://developer.ibm.com/aspera/docs/ibm-aspera-faspstream-sdk-3-8-0/. Note that using the SDK will require adhering to the public cloud documentation for the IBM Cloud Object Storage SDK Aspera service.

Edit the script to include your IBM cloud access credentials, bucket name, and other necessary information, then follow the rest of the instructions on page 7.

```
import os
import pprint
import urllib3
import uuid
import requests
import base64
import json
import sys
import os
import ibm_boto3
import datetime
import time
from ibm_botocore.config import Config
from cos_aspera import faspmanager2, transferspec
from cos_aspera.transferspec import TransferSpec, TransferSpecEncoder
from cos_aspera.faspmanager2 import startTransfer, configureLogLocation, ITransferListener

class TransferListener(faspmanager2.ITransferListener):
    def __init__(self, fnToCall):
        faspmanager2.ITransferListener.__init__(self)
        self.fnToCall = fnToCall
    def transferReporter(self, xferId, message):
        self.fnToCall()

def reporter():
    pass

def ats_remote_host_all_format(url):
    index = url.index('.')
    host = url[:index] + '-all' + url[index:]
    host = host.replace('https://', '')
    host = host.replace(':443', '')
    return(host)


transferReporter = TransferListener(reporter)
transferReporter.thisown = 0

# COS Related Vars
api_key = "[YOUR API KEY]"
bucket_name = "[YOUR BUCKET NAME]"
service_url = "s3.us-south.objectstorage.softlayer.net"
auth_endpoint = "https://iam.bluemix.net/oidc/token"
service_instance_id = '[YOUR INSTANCE ID]'

# Upload File Vars
source = "./file20gb.bin"
multi_session = 10
multi_session_threshold = 100000000
target_rate_kbps = 1000000

# Destination where file will end up in COS, modify as needed
destination = source.split('/')[-1]
```

```
# log location, must pre-create directory
local_log_path = "./logs"

new_session = ibm_boto3.Session()
client = new_session.client('s3',
                            ibm_api_key_id=api_key,
                            ibm_auth_endpoint=auth_endpoint,
                            ibm_service_instance_id=service_instance_id,
                            verify=False,
                            use_ssl=True,
                            config=Config(signature_version='oauth'),
                            endpoint_url='https://%s' % service_url)

# GET ?faspConnection info from COS staging
response = client.get_bucket_aspera(Bucket=bucket_name)
ats_endpoint = response['ATSEndpoint']
access_key_id = response['AccessKey']['Id']
secret_key = response['AccessKey']['Secret']

print('ATS: ' + ats_remote_host_all_format(ats_endpoint))

###############################################################################################
# Temp step take users API key, talk to IAM staging to get IAM cookie
###############################################################################################
json_data = {'grant_type': 'urn:ibm:params:oauth:grant-type:apikey',
             'receiver_client_ids': 'aspera_ats',
             'apikey': api_key,
             'response_type': 'delegated_refresh_token'}

headers = {'Content-Type': 'application/x-www-form-urlencoded',
           'Accept': 'application/json'}
response = requests.post(auth_endpoint, auth=('bx', 'bx'), data=json_data, headers=headers)

token = json.loads(response.content)['delegated_refresh_token']

# Starting to build headers for the ATS request later
credentials = {'type': 'token',
               'token': {"delegated_refresh_token": token}}

# ###############################################################################################
# # Contact ATS to setup transfer and get initial transfer spec (upload/download setup Rest API)
# ###############################################################################################
headers = {'X-Aspera-Storage-Credentials': json.dumps(credentials)}
auth = (access_key_id, secret_key)
data = {'transfer_requests': [{'transfer_request': {'paths': [{'destination': destination}],
        'tags': {'aspera': {'node': {'storage_credentials': credentials}}}}}]}

setup_transfer_response = requests.post(url=ats_endpoint + '/files/upload_setup', auth=auth,
headers=headers, json=data)
transfer_spec_json = json.loads(setup_transfer_response.content)
transfer_spec_json['transfer_specs'][0]['transfer_spec']['destination_root'] = '/'
transfer_spec_json['transfer_specs'][0]['transfer_spec']['paths'][0]['source'] = source
transfer_spec_json['transfer_specs'][0]['transfer_spec']['multi_session'] = multi_session
transfer_spec_json['transfer_specs'][0]['transfer_spec']['target_rate_kbps'] = target_rate_kbps
transfer_spec_json['transfer_specs'][0]['transfer_spec']['multi_session_threshold'] = multi_session_
threshold

# set the remote host to the 'xxxx-all.aspera.io' format, this allow multisession to transfer accross
ats nodes
transfer_spec_json['transfer_specs'][0]['transfer_spec']['remote_host'] = ats_remote_host_all_
format(ats_endpoint)

# ###################################################################
# # Send upload/download request via Aspera SDK method to start transfer.
```

```
# ######################################################################
xferId = str(uuid.uuid4())
print('xferid:' + xferId)
starttime = time.time()

json_body = json.dumps(transfer_spec_json)

pprint.pprint(json_body)

configureLogLocation(local_log_path)

print('transfer started...')
startTransfer(xferId, None, json_body, transferReporter)

while faspmanager2.isRunning(xferId):
    time.sleep(0.2)
    pass

faspmanager2.stopTransfer(xferId)
print('transfer stopped...')
endtime = time.time()
print(str(round(endtime-starttime, 2)) + 's')
```

Read the report at http://facts.pt/docm5vh ▶

This project was commissioned by IBM.