Going the distance

Copying data over high latency network links

Peter Boros
Principal Architect @ Percona

PERCONA

About me

- Principal Architect at Percona
- Focused on automation and performance tuning
- Among others, worked at Dropbox, Zuora, Sun microsystems

Agenda

- Long distance copy: What is the difference?
- Measurement setup
- Some TCP/IP
- Benchmarking
- Parallel TCP streams
- Copying an existing backup
- Streaming backups

Long distance copies

What? Why?

- Long distance means more latency
  - Not necessarily less bandwidth

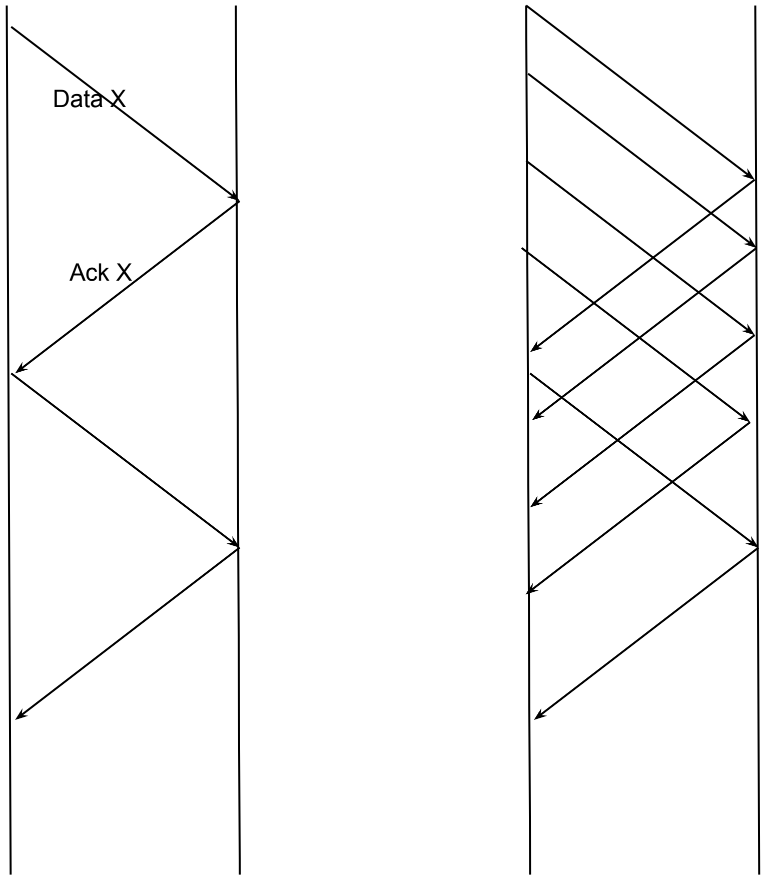
- Disaster recovery purposes
  - Data in distant environment: we need initial copy
  - This may be repeated through the lifecycle of the DR environment
- Moving data to the cloud or between cloud providers
- Disaster recovery testing (practice exercises)
- Read replicas in remote regions

First: measure

Measurement setup

- Actual databases or data are not needed to validate the methods
- Used AWS
  - This discussed here are general
- Various instance types in the same region (us-west-2)
- Various instance types between 2 distant regions (eu-central-1)
- The problem itself is not database related
- Tested with t2.micro instances
  - Results are reproducible in the free tier
  - Larger instances will have more consistent speeds

Some theory: TCP window scaling



- By default, TCP is not great over high latency links
- Sliding window mechanics of TCP are here to help
- Sending the next packet doesn't need to wait for the acknowledgment
- Selective acknowledgement (sack) helps to acknowledge multiple packets with a single answer
- Adjusted dynamically

PERCONA

Ubuntu 20.04 defaults

```
net.core.wmem_default = 212992
net.core.wmem_max = 212992
net.ipv4.tcp_wmem = 4096   16384  4194304

net.core.rmem_default = 212992
net.core.rmem_max = 212992
net.ipv4.tcp_rmem = 4096   131072 6291456
net.ipv4.udp_rmem_min = 4096

net.ipv4.tcp_window_scaling = 1
```

Default iperf same region

```
# iperf3 -s -p 9001
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------


# iperf3 -c 1.2.3.4 -p 9001
…
[  5]   0.00-10.00  sec  1.03 GBytes   883 Mbits/sec  4698
sender
```


PERCONA

Same region, but limiting the window size

```
# iperf3 -s -p 9001
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------


# iperf3 -c 1.2.3.4 -p 9001 -w 1400
…
[  5]   0.00-10.00  sec  18.2 MBytes  15.3 Mbits/sec
receiver
```


PERCONA

Promising!

Different (us-west-2, eu-central-1)

```
# iperf3 -s -p 9001
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------
```

```
# iperf3 -c 1.2.3.4 -p 9001
…
[  5]   0.00-10.14  sec  77.7 MBytes  64.3 Mbits/sec
receiver
```

Some tuning for high latency

```
net.core.wmem_max = 33554432
net.core.rmem_max = 33554432
net.ipv4.tcp_rmem = 10240 87380 33554432
net.ipv4.tcp_wmem = 10240 87380 33554432
net.core.netdev_max_backlog = 5000
```

Different (us-west-2, eu-central-1)

```
# iperf3 -s -p 9001
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------
```

```
# iperf3 -c 1.2.3.4 -p 9001 -w 8388608
…
[  5]   0.00-10.14  sec  83.1 MBytes  68.8 Mbits/sec
receiver
```

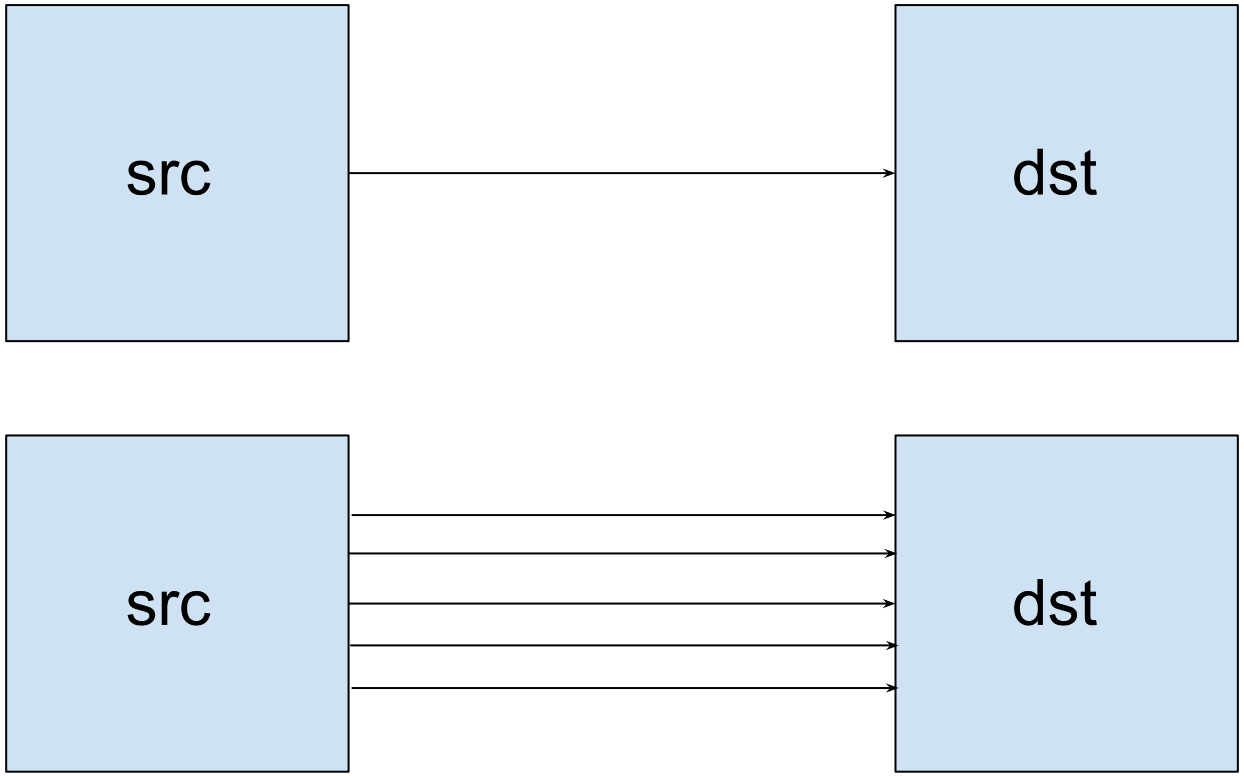
Not great results

- **Slight but consistent difference**
- **Requesting a larger windows at the iperf level doesn't make much difference**
- **We already had**
  - net.ipv4.tcp_sack = 1
  - net.ipv4.tcp_window_scaling = 1
- **Tunables are available on a per connection basis**
  - Several applications support it (for example bbcp)

PERCONA

Parallelism

Single vs multiple streams



Different (us-west-2, eu-central-1)

```
# iperf3 -s -p 9001 -P 4
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------
```

```
# iperf3 -c 1.2.3.4 -p 9001
…
[  5]   0.00-10.14  sec  83.1 MBytes  68.8 Mbits/sec
receiver
[SUM]   0.00-10.14  sec   254 MBytes   210 Mbits/sec
receiver
```


PERCONA

Different (us-west-2, eu-central-1)

```
# iperf3 -s -p 9001 -P 6
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------
```

```
# iperf3 -c 1.2.3.4 -p 9001
…
[  5]   0.00-10.14  sec  83.1 MBytes  68.8 Mbits/sec
receiver
[SUM]   0.00-10.14  sec   383 MBytes   317 Mbits/sec
receiver
```


PERCONA

Different (us-west-2, eu-central-1)

```
# iperf3 -s -p 9001 -P 16
-----------------------------------------------------------
Server listening on 9001
-----------------------------------------------------------
```

```
# iperf3 -c 1.2.3.4 -p 9001
…
[  5]   0.00-10.14  sec  83.1 MBytes  68.8 Mbits/sec
receiver
[SUM]   0.00-10.14  sec   578 MBytes   478 Mbits/sec
receiver
```

Parallel TCP streams

- Different source port for each stream
- Not necessarily different destination port for each stream
  - Depends on the implementation
  - With one destination port, the listener needs to handle IO multiplexing

Parallel streams is the way to go!

PERCONA

Can be useful even locally

- **Modern, high performance network controllers**
  - Can’t be saturated with a single stream
  - Have multiple interrupt channels for both TX and RX

Copying an existing backup

Copying existing backup

- Have a set of files to copy
- Want to copy them using multiple TCP streams
- Normal methods could be scp, tar | nc, all single streamed

bbcp

- Does exactly this
- Using SSH for control channel
- Seems like SCP, but it’s not
- Control traffic is encrypted, data is not!

bbcp setup (Ubuntu 20.04)

```
sudo apt-get install libssl-dev build-essential zlib1g-dev git
git clone https://www.slac.stanford.edu/~abh/bbcp/bbcp.git/
cd bbcp/src
make
sudo cp ../bin/amd64_linux/bbcp /bin/bbcp
bbcp --version
```

bbcp example

```
bbcp \
-P 16 \
-Z 9001:9016  -r testdir ubuntu@dest_machine:/home/ubuntu/
```

Caveats!

- Doesn’t handle ~ (it’s like scp but it’s not)
- The bbcp binary must be in the path of the receiving machine
- Bi-directional communication is needed (receiver connects back to sender)
- Data is not encrypted

Parallel xtrabackup

PERCONA

Parallel xtrabackup

- **xbstream will emit a single stream that can be copied**
- **nc, socat and the likes are using a single stream**
  - will be inefficient on high latency links
- **network copy if often the bottleneck**

PERCONA

Out of the box: xbcloud and object storage

xbcloud

- **xbstream will emit a single stream that can be copied**
- **nc, socat and the likes are using a single stream**
  - will be inefficient on high latency links
- **network copy if often the bottleneck**
- **xbcloud to the rescue**
  - copy first to the object storage, copy within the object storage to another region
  - both can be parallel

PERCONA

xbcloud example

```
$ xtrabackup --backup --stream=xbstream --parallel=10
--extra-lsndir=/tmp --target-dir=/tmp | \
xbcloud put --storage=s3 \
--s3-endpoint='s3.amazonaws.com' \
--s3-access-key='YOUR-ACCESSKEYID' \
--s3-secret-key='YOUR-SECRETACCESSKEY' \
--s3-bucket='mysql_backups'
--parallel=10 \
$(date -I)-full_backup
```

s3 region copy example

```
$ aws s3 cp s3://src-bucket-region-1/ \
      s3://target-bucket-region-2/ \
      --recursive \
      --source-region region-1 \
      --region region-2 \
      --max-concurrent-requests=50
```

Summary

- Use xbcloud to copy to object storage
- Copy the data to another region of the object storage
  - Or specify the remote region for xbcloud
- Restore locally from the target object storage
- The example was for AWS and S3, but xbcloud works for other object storage too
- You will get the high throughput as you would get with bbcp

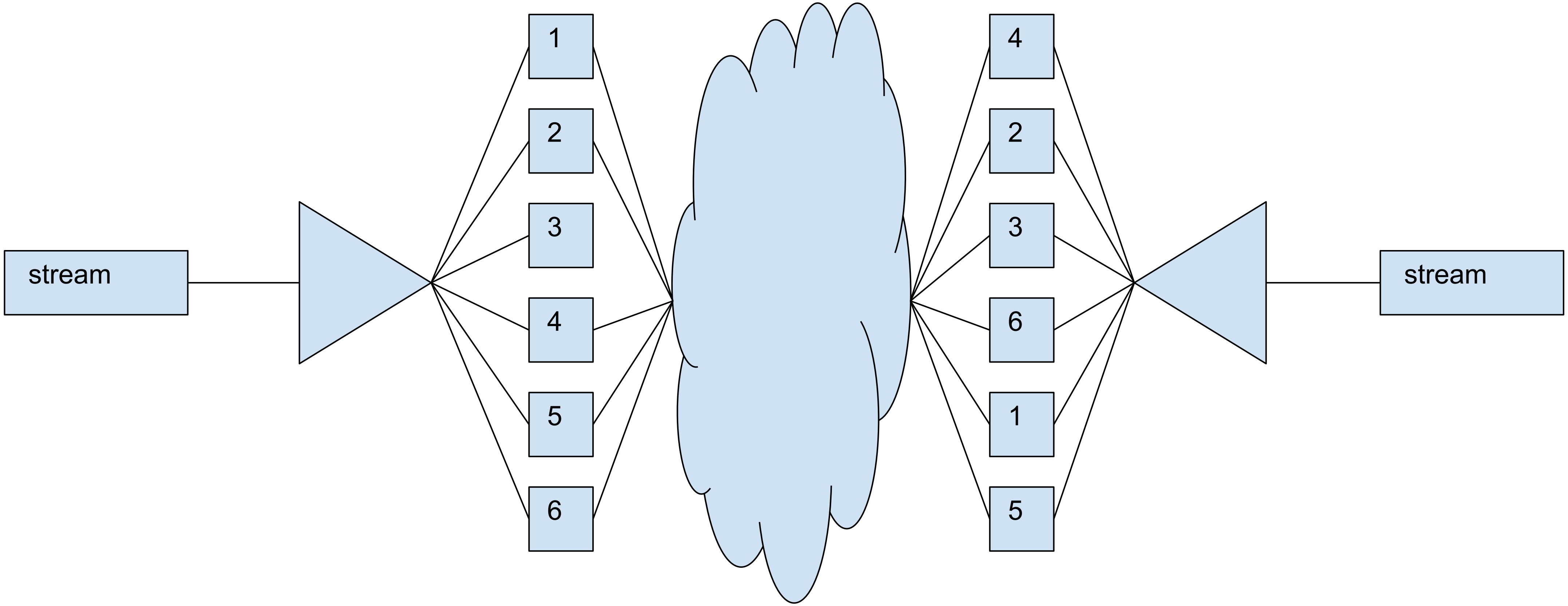
Reading a stream in chunks

Reading the stream in chunks

- **How does it work part**
- **A stream can be read in chunks locally**
- **The chunks can be processed in parallel**
  - Sending over the network
  - Compression
  - Encryption
  - Anything expensive
- **Tools mentioned earlier have similar mechanics**

PERCONA

Reading the stream in chunks



Reading the stream in chunks

- **No out of the box solution for it**
- **A stream can be read in chunks locally**
- **The chunks can be processed in parallel**
  - Sending over the network
  - Compression
  - Encryption
  - Anything expensive

PERCONA

Simple python example

```
In [1]: import subprocess

In [2]: class DataChunk(object):
   ...:     def __init__(self, data, seqno):
   ...:         self.data = data
   ...:         self.seqno = seqno
   ...:
   ...:     def __repr__(self):
   ...:         return "DataChunk({seqno})".format(seqno=self.seqno)
   ...:

In [3]: chunks = []

In [4]: xb_proc = subprocess.Popen(["xtrabackup", "--backup", "--stream=xbstream"],
   ...: stdout=subprocess.PIPE, stderr=subprocess.PIPE)

In [5]: chunks.append(DataChunk(xb_proc.stdout.read(64*1024*1024), 1))

In [6]: chunks.append(DataChunk(xb_proc.stdout.read(64*1024*1024), 2))

In [7]: chunks
Out[7]: [DataChunk(1), DataChunk(2)]

In [8]: len(chunks[0].data)
Out[8]: 67108864
```

Thank you!

Q&A

PERCONA