Optimised Data Transfer with Alex

Hadrien Reynaud - 06/02/2024

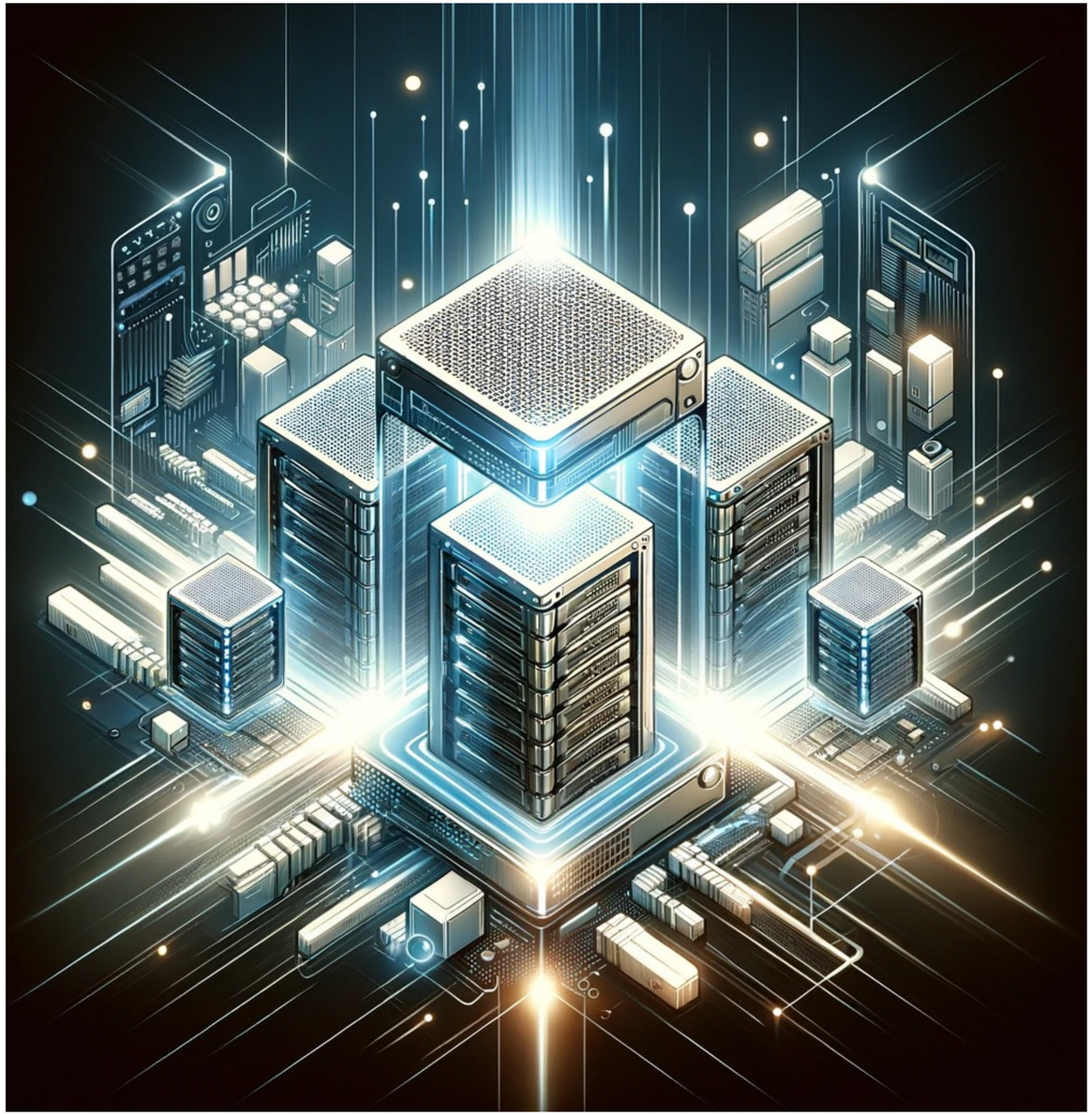
$TMPDIR

+ Fast read/write

+ Not bottlenecked by the network speed

- Only exists for the duration of the job

- Moving data takes time from the total execution time (ie. loss of compute)



Solution 1 - Fast Data Transfer

Goal

- We want to use `rsync` to copy data over `ssh`
- One `rsync` process copies data at 10Mo/s - 20Mo/s
- The max bandwidth between FAU servers and Alex is ~25Gb/s*

- **How do we maximise bandwidth usage ?**
- Move the data using parallel processes of `rsync`, using `xargs`
- Goal: move 500Go in ~40 minutes

*Based on the theoretical link speed between both servers

Solution 1 - Fast Data Transfer

Pre-requisite

- Data must be stored on a FAU server that can connect to Alex over `ssh`
- Requires setting up the `ssh` config on both Alex and the “local” machine

1. Password less login to allow parallel calls
2. Use aliases for ease of use

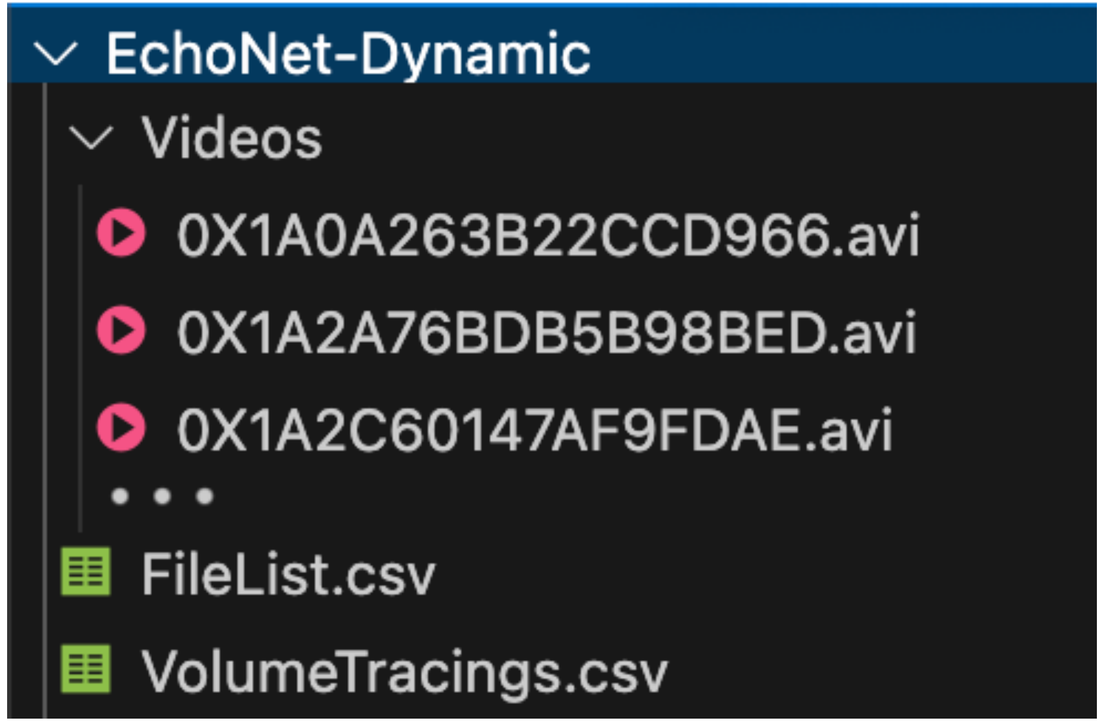
```
# On the "local" machine
Host alex
        HostName alex.nhr.fau.de
        User b180dc18
        IdentityFile ~/.ssh/id_rsa

# On Alex
Host harendotes # <-- name of the local machine
        HostName idea-harendotes.aibe.uni-erlangen.de
        User at70emic
        IdentityFile ~/.ssh/id_rsa
```

Solution 1 - Copying data faster

Data structure

- Data must be contained in one folder (with any amount of subfolders)
- No limitation on data types, structure, etc…



Solution 1 - Copying data faster

Data compression on host machine

- The data is (1) zipped and (2) split into 100Mo chunks
- The 100Mo chunks are optimal for fast transfer: not to small, easy to parallelise

```
cd /parent/of/data/folder/

# create zip archive with 0 compression (faster unzip)
zip -0 -r data.zip data/

mkdir data_parts

# split the zip file into 100Mo chunks
split -b 100M data.zip data_parts/part_
```

Solution 1 - Copying data faster

Data pulling in a slurm job

- Retrieve the list of 100Mo parts to transfer
- Call `rsync` with `xargs` on that list

```
# Prepare reception folder on local node disk
mkdir -p "$TMPDIR/data_parts"

# Path to the data on the FAU server
remote_data="/path/to/data_parts"

# Get list of data parts
list=$(ssh harendotes "find '$remote_data' -name 'part_*'")

# Copy all parts, using 16 parallel processes
echo "$list" | xargs -I {} -P 16 rsync -az harendotes:"{}" "$TMPDIR/data_parts"
```

Solution 1 - Copying data faster

Data reconstruction

- Un-split the zip
- Unzip the data
- Data is ready

```
# merge all the parts back into a zip file
cat $TMPDIR/data_parts/part_* > "$TMPDIR/data.zip"

# unzip the data locally
unzip -q "$TMPDIR/data.zip" -d "$TMPDIR/data/"
```

Solution 1 - Copying data faster

Limitations

- The host machine must have a high-bandwidth connection to Alex
- By default, host machines limit the number of concurrent `ssh` connections to 16
  - If multiple user or multiple experiments pull from the same host, `rsync` will fail and data will not be copied, failing the whole job.
  - It is possible to change the `ssh` server configuration to increase that limit (requires `sudo`)
- Your computation cannot start until the data is downloaded, which reduces your compute time (given the wall time of 24h)
- This approach is interesting if the data transfer time remains low (few hours)

Solution 2 - Background Data Stream

Goal

- Start training immediately, without waiting for any data transfer

- Use `webdataset` to stream the data from any* location
- Reduce compute idle time (=data transfer) to a minimum
- Allows to move 25Tb in ~8 hours (split over 8 nodes)

https://github.com/webdataset/webdataset

*Assuming you have access and admin rights / some control over the machine where the data is located

Solution 2 - Background Data Stream

Pre-requisite

- Data must be stored on a server reachable by Alex (req. proxies)
- The server must be ready to send large amounts of data (req. file server)
- Data must be formatted in the right structure (req. pre-processing)

```
# allow internet access
export http_proxy=http://proxy.rrze.uni-erlangen.de:80
export https_proxy=http://proxy.rrze.uni-erlangen.de:80

# allow local (FAU) servers access
export no_proxy="localhost,127.0.0.1,10.76.21.13,10.76.21.10"
export NO_PROXY="localhost,127.0.0.1,10.76.21.13,10.76.21.10"
```

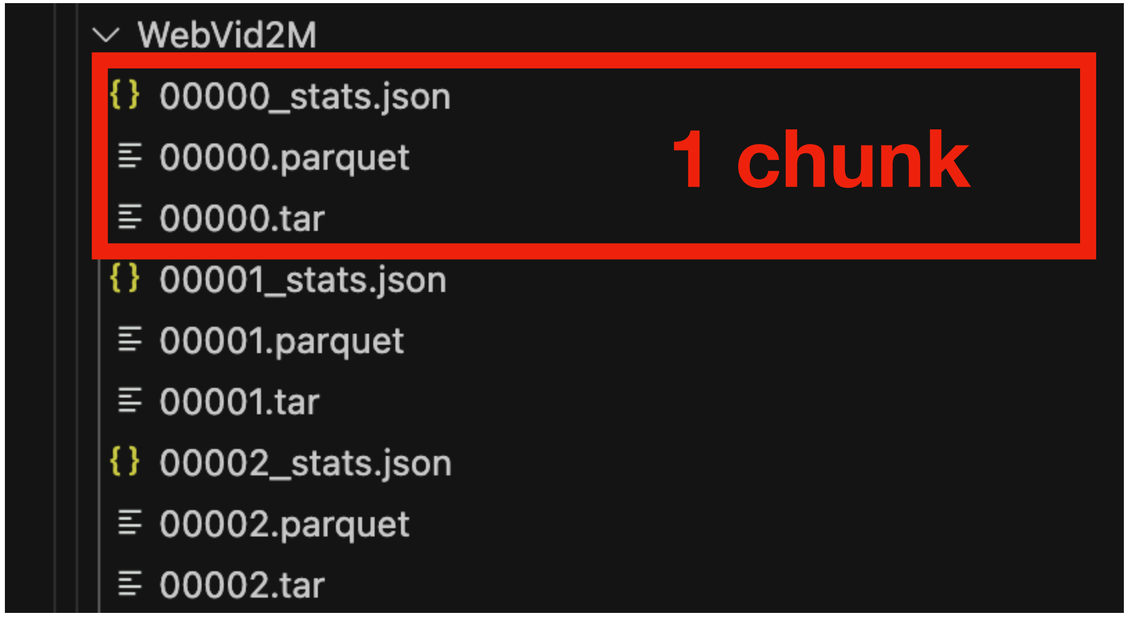
Solution 2 - Background Data Stream

Data structure

- The data must be organised into `tar` files and the labels (class, text, etc) into `json/txt` files.

1. Split data into ~1Go chunks (ex. group of images + 1 `json` file)
2. Data chunks are archived into a `tar` file. Grouped data share their name, ex: `00001.tar` and `00001.json`

- This may require lots of pre-processing.



Solution 2 - Background Data Stream

Data serving

- Need to setup a server to send the properly structured data
- Simple solution: NGINX server (python server not powerful enough)

```
sudo apt install nginx
sudo systemctl start nginx
sudo nano /etc/nginx/sites-available/default

# In nano
server {
    listen [::]:80 default_server; # set the connection port
    root /path/to/data/folder; # set the path to the data
    [...]
}
# ctrl + s, ctrl + x to save and exit

sudo nginx -t # check that the configuration works
sudo systemctl reload nginx
```

Solution 2 - Background Data Stream

Data pulling in a slurm job

- Switch the usual dataset class for a WebDataset and WebLoader

```
import webdataset as wds

chunks = "http://10.XX.XX.XX/data/{00000..00543}.tar"

# Create the datasets over the chunks
trainset = wds.WebDataset(chunks, resampled=True, cache_dir=None, shardshuffle=True)

# Configure sample shuffling, decoding and transforms.
trainset = trainset.shuffle(1000).decode("pil").map(custom_func)

# Batch data in dataset, NOT dataloader
trainset = trainset.batched(64)
```

Solution 2 - Background Data Stream

Data pulling in a slurm job

- Switch the usual dataset class for a WebDataset and WebLoader

```
# WebLoader is PyTorch DataLoader with some convenience methods.
trainloader = wds.WebLoader(trainset, batch_size=None, num_workers=4)

# Unbatch, shuffle between workers, then rebatch.
trainloader = trainloader.unbatched().shuffle(1000).batched(64)

# Since we are using resampling, the dataset is infinite; set an artificial epoch size.
trainloader = trainloader.with_epoch(100_000)

for batch in trainloader:
    image, label = [batched_output_of_custom_func]
    ...
```

Solution 2 - Background Data Stream

Bonus

- WebDataset supports caching the data locally - avoids pulling the same data multiple times ($TMPDIR is usually 8Tb per node).
- `custom_func` is an on the fly pre-processing function, can handle all sorts of processing.
- See https://github.com/webdataset/webdataset to learn more.

Solution 2 - Background Data Stream

Limitations

- Heavy setup cost (data pre-processing, http server, new data loading)
- Requires a dedicated http file server (can be hosted remotely, ex. AWS, GCP)
- Worth the effort only for large datasets (3Tb+)

Recap

Parallel rsync vs WebDataset

`rsync + xargs`	WebDataset
+ Simple to setup	+ Computation can start immediately
+ Easy to modify / adapt	+ Scales infinitely
+ Fits all datasets	- Long / difficult to setup
- Blocks the job until data is transferred	- May not fit all datasets
• Best for <2Tb datasets (<3h transfer)	• Best for 2Tb+ datasets

Thank you