

---

# Data Storage & Access



**Open Science Grid**

---

OSG Summer School  
July 27-31, 2015

---



# How big is Big?

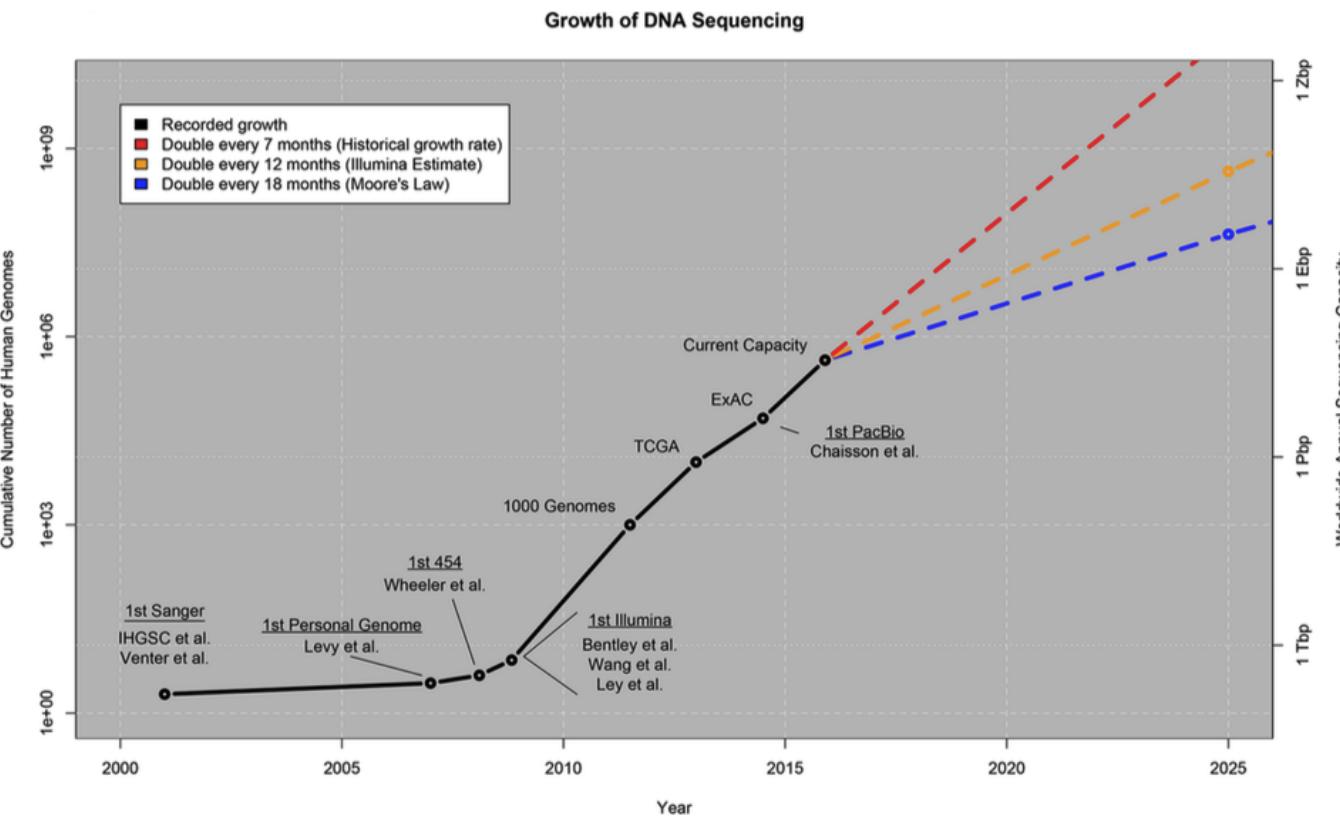
# Some examples of Big

---



- A single human genome
  - 200GB [1]
- Sloan Digital Sky Survey
  - 116 TB in Data Release 12 [2]
- ATLAS experiment at the LHC
  - 140 PB on disk in 2014 [3]
- Facebook's data warehouse in 2014
  - 300 PB [4]

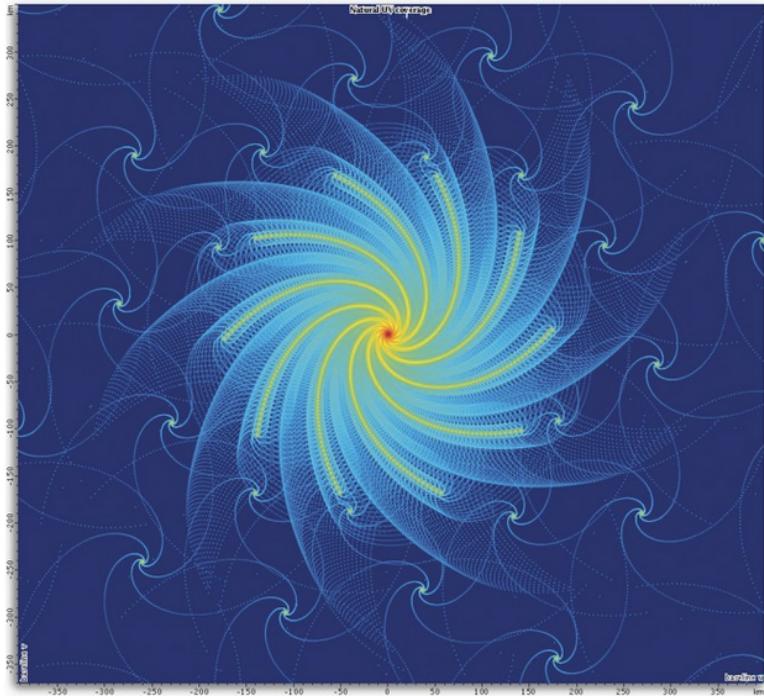
# Genomics - projected 10 year growth



> 1 Ebp/year  
(base pairs) ⇒  
1-40 EB/year

PLOS Biology |  
DOI:10.1371/journal.pbio.1002195  
July 7, 2015

# Square Kilometer Array ([www.skatelescope.org](http://www.skatelescope.org))



7 EB/y to disk in 2022 projected  
(~ 1 EB/day raw!)

<http://arxiv.org/pdf/1101.1355.pdf>

# Storage systems

---



- Most university HPC centers provide storage to users, at various costs depending on type
  - Cheaper, slower → high capacity
  - Other applications may require fast (e.g. SSD) storage
  - Tape systems can be used for archival, but rarely within live computational workflows
- National cyberinfrastructure centers also offer storage with varying characteristics and costs
- AWS storage - S3



- What good is all of this “big data” if you can’t move it around?
- Many things to think about:
  - How much data does my application need?
  - How fast can I move it?
  - How quickly can my application consume it?
  - How to access my data on a remote worker?
  - What changes as I scale up?



# Getting started

# Data transfer basics

---



- Standard tools like scp, rsync work well for moving data to submit nodes
  - Does **not** (usually) work inside a job!
- Also consider: Where to put data?
  - /home often has quotas in place.
  - Usually some dedicated storage in place
  - If in doubt, ask your system operator.

# Dealing with larger datasets

---



- Many datasets are too large to keep a local copy
- How to move these around?
  - We recommend Globus
    - Web-based platform for moving data between facilities or workstations
  - Access via <http://portal.osgconnect.net> if using OSG Connect

# Moving data with Globus

---

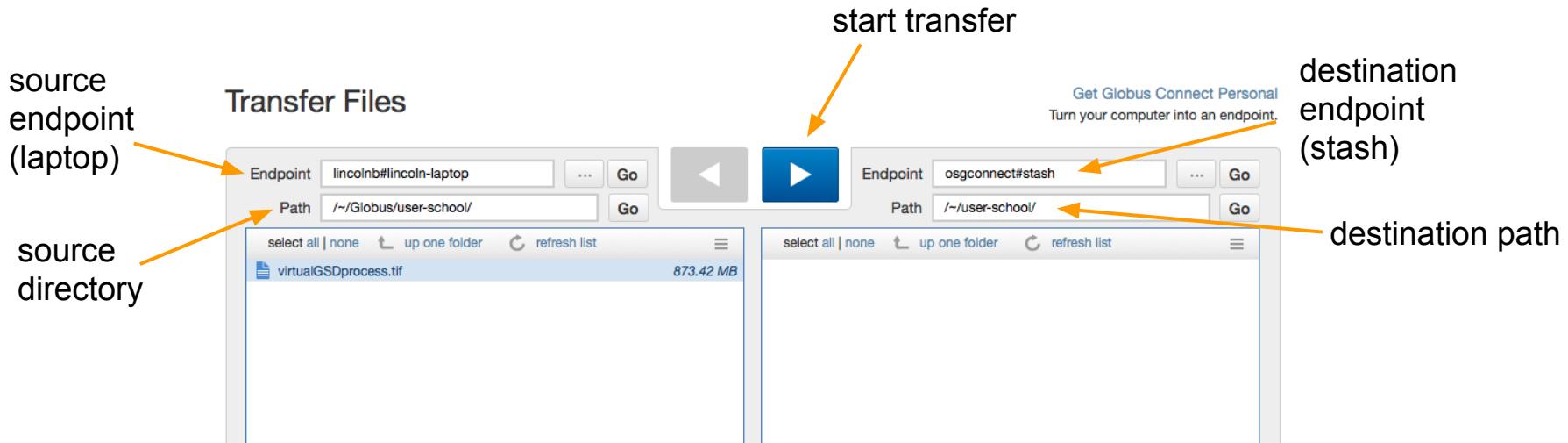


- Move data to/from:
  - OSG Connect (Stash)
  - XSEDE or Blue Waters HPC centers
  - your local research computing facility
  - local resources (e.g., laptop)
- Restarts failed/incomplete transfers, manages data integrity, designed for high-throughput transfers

# Globus Online Example



## ● Transferring files from laptop to Stash



# Transferring data to a job

---



- OSG does not have a shared filesystem!
  - Jobs must transfer their input files to the remote worker.
- scp, rsync, etc typically will not work here
- Basic mechanism: HTCondor file transfer
  - `transfer_input_files = ...`
  - Can also recursively copy directories, transfer via other protocols such as HTTP (see later)

# HTCondor File Transfer Example

---



- Print the contents of hello\_world.txt:

```
universe = vanilla
```

```
output = job.out
```

```
error = job.err
```

```
log = job.log
```

```
transfer_input_files = hello_world.txt
```

```
executable = /bin/cat
```

```
arguments = hello_world.txt
```

```
queue
```

# Getting data back out

---



- HTCondor monitors the job sandbox.
  - Files created within the sandbox will get transferred back out to the user.
- Can also explicitly create a list of outputs with `transfer_output_files`
  - Useful when working on scratch disk

# I/O Considerations

---



- Bandwidth is not infinite!
- Most workers connected at 1Gb/s (125MB/s)
- Realistically expect to copy data at rates of 1MB/s to 50MB/s
  - May be more or less depending on disk, local network, filesystem configuration, # of concurrent transfers, and external network

# I/O Considerations

---



- Disks tend to be anywhere from 40-80MB/s
  - Varies for sequential vs random reads
  - Remember: shared resource!
- How fast can your application consume data?
  - e.g., typical ATLAS analysis job processes data at approximately 40MB/s
  - No good rule of thumb, experiment and find out.

# File size vs number of files

---



- Two important dimensions of a transfer task:
  - Average file size and the number of files
- Is one 1GB file the same as  $10^6$  1KB files? **No!**
- The difference is in the filesystem metadata
  - a million 1KB files -> a lot of overhead
  - gets worse as you start moving them around
    - overhead for the transfer services

# Dealing with small files

---



- We know that large amounts of small files are bad
- How to handle this?

# Dealing with small files

---



- We know that large amounts of small files are bad
- How to handle this?
  - Archive (zip/tar) small files.
- Compression saves a lot of time with small performance penalties
- Tools like zcat (1) can read out of compressed files on the fly

# File compression how-to

---



- Use the tar utility:

```
$ tar -cvzf my_file.tar.gz my_directory/
```

*create new tar file*

*show progress*

*use gzip  
compression*

*operate on a file/directory*

*standard practice*

```
my_directory/
```

*file(s) to compress*

# Compression example

---



- Wikipedia public data dump.
  - Contains all articles and changes to articles
- 168 compressed (7z) XML files
  - One compressed file: 127 MB
  - Same file, uncompressed: 14 GB
- ~110:1 compression ratio!
  - Not all files will compress this well

# Scratch disks

---



- In the previous example, decompressed file was 14GB
  - Put this in scratch if available!
  - Scratch location: \$OSG\_WN\_TMP
- Why scratch?
  - Job sandbox is often restricted in size
  - Scratch is usually on separate/faster disks

# More on scratch

---



- Scratch won't necessarily be cleaned automatically
  - clean up after yourself.
- \$OSG\_WN\_TMP also not necessarily defined
  - Need a little if...then...else in your wrapper

# Scratch example

---



```
#!/bin/bash

if [ -n $OSG_WN_TMP ]; then
    MY_SCRATCH=$(mktemp --tmpdir=$OSG_WN_TMP XXXXX)
else
    MY_SCRATCH=$(mktemp --tmpdir=$(pwd) XXXXX)
fi

... (move data to $MY_SCRATCH, do work) ...

rm -rf $MY_SCRATCH
```



- Exercise 1
  - Refresher of scp, rsync,
  - introduction to Globus
- Exercise 2
  - Transferring files in and out using HTCondor's built-in file transfer



# Caching infrastructures



- Remember what is a good fit for OSG:
  - Jobs lasting 1-12 hours
  - Less than 2GB of memory
  - Input/out less than 10GB
- Data needed per job..
  - can influence all of the above
  - is determined on a case-by-case basis



- Would a single job with 100GB of input data be a good fit for OSG?



- Would a single job with 100GB of input data be a good fit for OSG?
- Assume the worst: 100GB / 1MB/s
  - Over a day just to transfer data
- Can we improve this job?
  - Does the data need to be read in its entirety, or just parts?
  - Can we compress the data?



- HTTP
- Caching / StashCache
- Remote I/O
- Pre-staging / OSG Storage Element

# More on HTCondor File Transfer

---



- When using `transfer_input_files`, the data goes through the submit node
  - Potentially a bottleneck
- Suppose you have 1000 jobs starting simultaneously, each transferring 1GB input files. If the submit node has a 1Gbps connection, how long will it take?
  - (Hint: 1Gbps = 125MB/s; 1GB = 1024MB)

# HTCondor File Transfer Plugins

---



- HTCondor also supports various plugins for file transfer
- To use the HTTP plugin, for example, simply:
  - `transfer_input_files = http://<path to file>`
- Takes the network load off of the submit host
  - Scalability problems moved to web server developers/operators :)

# Simple HTTP plugin example

---



- Example that prints the contents of ‘hello.txt’:

```
universe = vanilla

output = job.out
error = job.err
log = job.log

transfer_input_files = http://s3-us-west-2.amazonaws.com/osg-user-school/hello.
txt
executable = /bin/cat
arguments = hello.txt

queue
```

# More on scaling up

---



- Do you transfer the same input data with each job?
  - Two options: caching or remote staging

# Caching and CDNs

---



- Imagine you own a popular website like YouTube, Netflix, Reddit, or Facebook
- Users are complaining that their videos load very slowly
- How would you get files to your users faster?

# Caching and CDNs

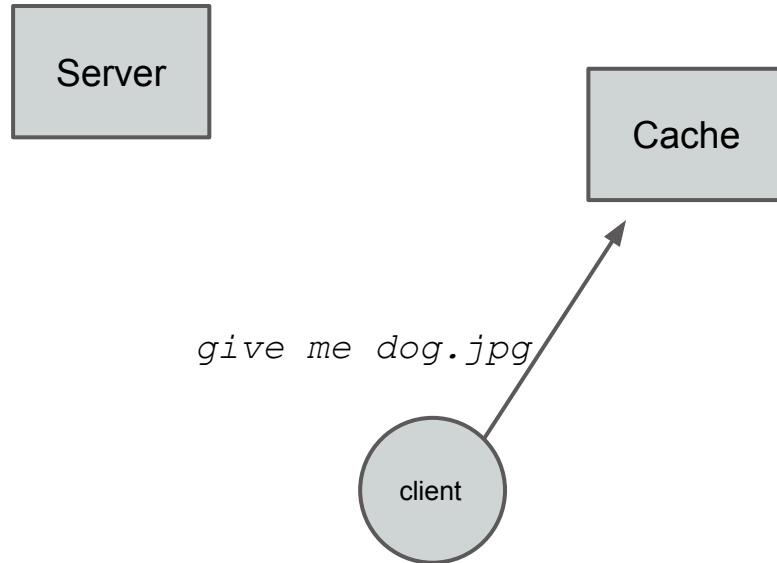
---



- Build a distributed caching infrastructure
  - Frequently accessed files stored on nearby caches
  - Each cache is as fast or faster than the source server
  - Source server only talks to the caches, not the end-users

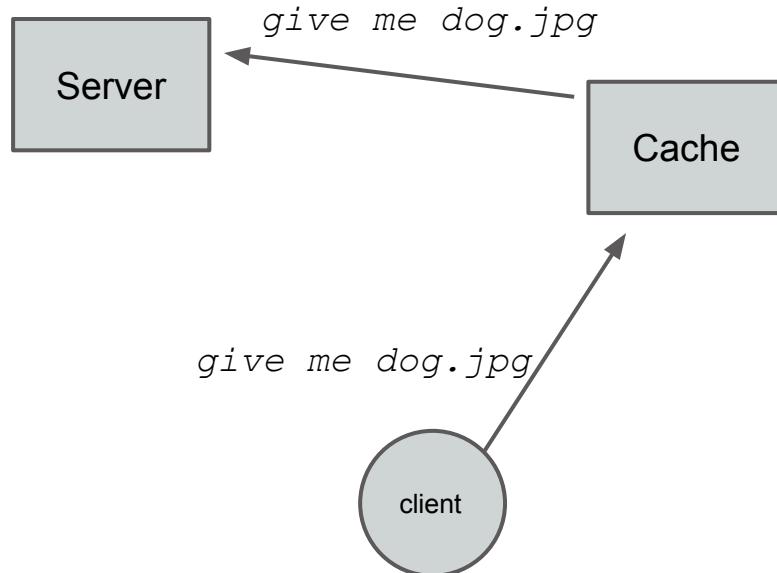
# Caching example

---



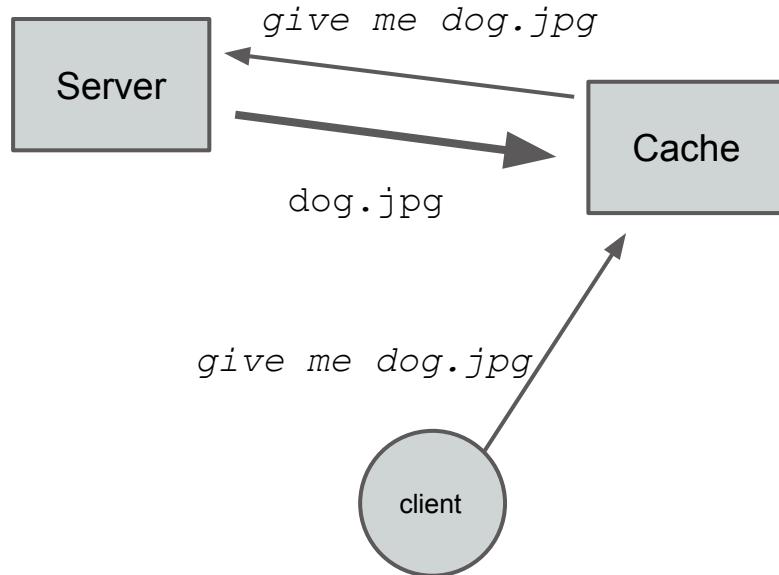
# Caching example

---

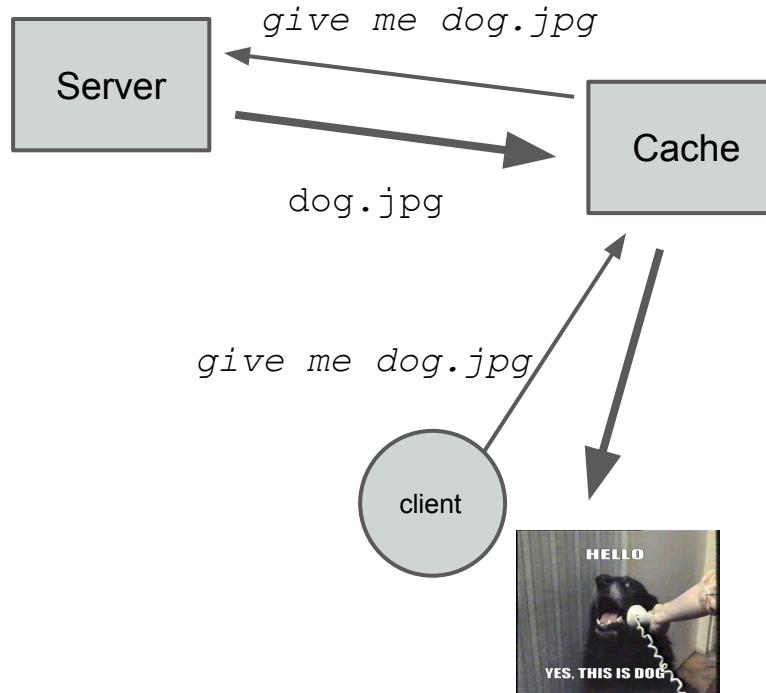


# Caching example

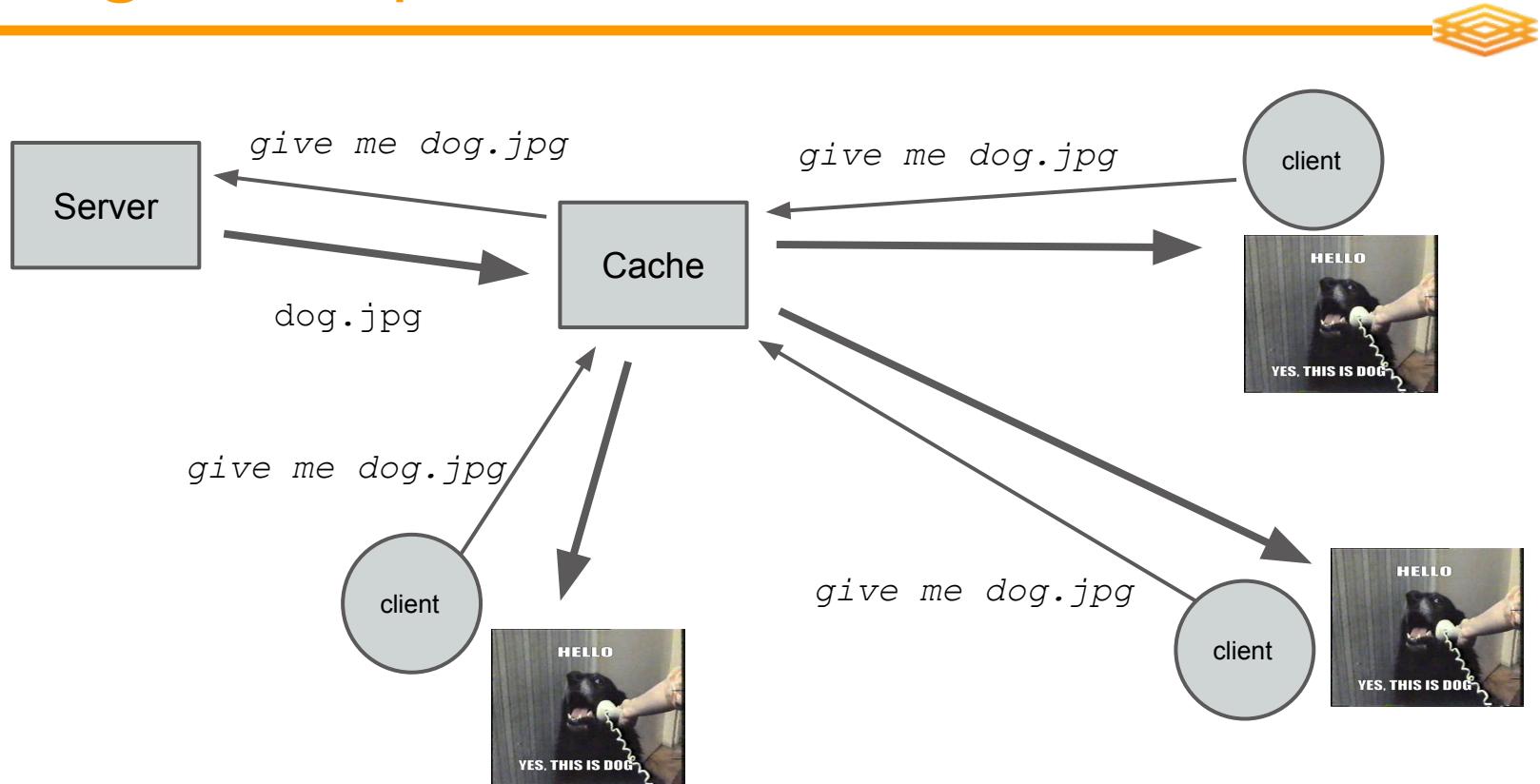
---



# Caching example



# Caching example



# HTTP Caching on OSG

---



- We can use the same technology to speed up data access
- OSG sites have local Squid servers for this reason
- Effective if:
  - You need the same dataset frequently
  - Cached files are reasonably small
    - Why?

# Using HTTP caches on OSG

---



- To access data via cache, need to set the environment variable `http_proxy`
- curl/wget tools automatically respect `http_proxy` if set
- Most OSG sites have a pointer to their local cache via `OSG_SQUID_LOCATION`

# Cache usage example

---



- Example script using wget

```
#!/bin/bash  
export http_proxy=$OSG_SQUID_LOCATION  
  
wget http://stash.osgconnect.net/+dbala/random_words
```

# StashCache

---



- Caching technology being developed by OSG for staging in larger datasets
  - 1GB - 1TB dataset sizes
- Files cached can be orders of magnitude larger than those in HTTP caches
- Designed with the opportunistic user in mind
- HTCondor transfer plugin and direct file access will be available.

# Exercises

---



- Exercise 3
  - Using HTTP and caching on OSG

---

# Remote I/O and the OSG Storage Element



# Dealing with *really* large datasets

---



- In some cases, the input dataset is too large to either:
  - transfer in a reasonable amount of time
  - store on a worker node.
- How do we deal with such datasets?
  - Two techniques: Split the data up, or use remote I/O

# Technique 1: Splitting up data

---



- No general recipe for this, some tips:
  - Unstructured text files can be manipulated simply with the standard UNIX tools (awk, sed, cut, etc)
  - Tar files can be extracted and reorganized
  - Tools like ImageMagick can split images

## Technique 2: Remote I/O

---



- Remote I/O allows a job to read remote data locally
- Either needs to be built into the protocol (e.g., XRootD) with a supporting client application (e.g., ROOT)
- Or user applications have to be tricked into thinking remote files are actually local
  - via trapping & redirecting system calls

# Tools for remote I/O

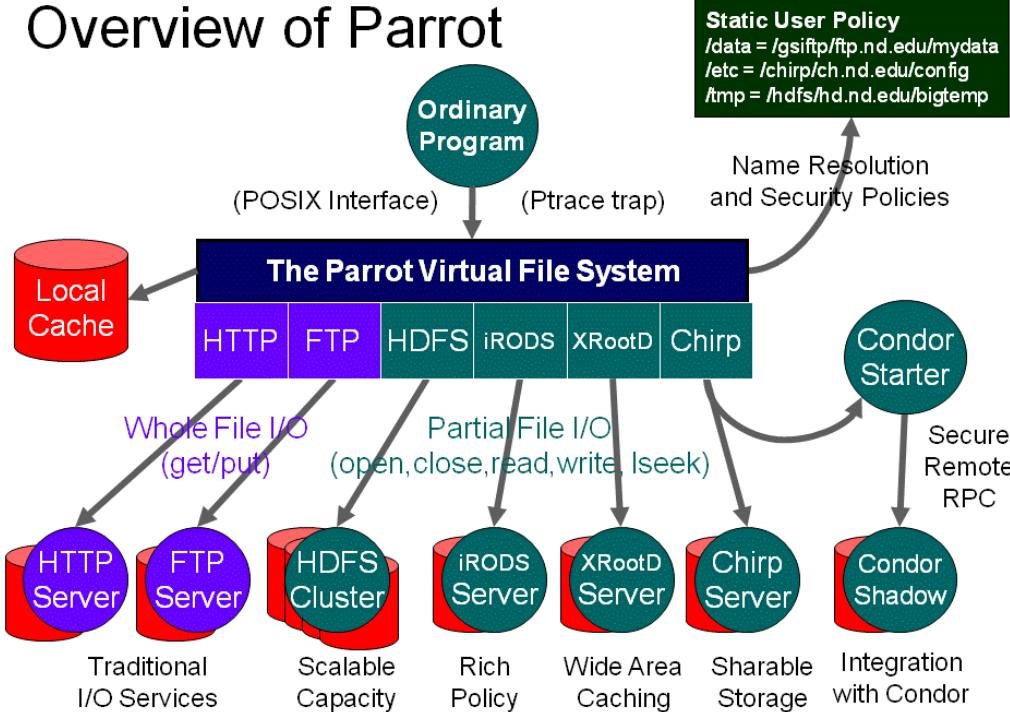
---



- Parrot
  - Intercepts system calls and rewrites them on the fly via the kernel debugging interface (ptrace)
- Chirp
  - Lightweight I/O protocol used by Parrot
  - Also available via condor\_chirp (see exercise)

# Parrot Diagram

## Overview of Parrot



*Diagram shamelessly stolen from: <http://ccl.cse.nd.edu/software/parrot/>*

# Parrot and Chirp caveats

---



- Does not work with some workloads
- Rewriting system calls on the fly can affect performance

# condor\_chirp example

---



# The OSG Storage Element (SE)

---



- The traditional OSG storage solution
- Jobs and data co-located at the same OSG site
- Multi-TB to multi-PB of storage space
- Works well for large collaborations (CMS, ATLAS, etc)
- Typical protocols: SRM, GridFTP, XRootD

# The OSG Storage Element (SE)

---



- Doesn't work so well for opportunistic use
- Why?
  - Need to know *a priori* where jobs will go in relation to data
  - No auto-expiration of old data

# A whirlwind tour through GridFTP/SRM

---



- GridFTP
  - extends the venerable File Transfer Protocol (FTP)
  - adds support for X509 authentication, fault tolerance, parallel transfers, etc

# A whirlwind tour through GridFTP/SRM

---



- Storage Resource Manager (SRM)
  - sits on top of GridFTP
  - adds quotas (space tokens), manages pools of GridFTP servers, etc

# When should I use the standard SE?

---



- SE currently most effective for staging large (multi-GB to TB) amounts of data in/out of a site
- However:
  - Using an SE requires an X509 certificate and some special setup.
  - If you think this applies to you, come talk to us.

# Summary

---



- Many ways to access data on OSG
  - HTCondor's built-in file transfer, external protocols (e.g. HTTP), caching, remote I/O
- The “best” access method will depend on your application requirements.

# References

---



- [1] <http://recode.net/2015/04/22/the-third-phase-of-big-data/>
- [2] [http://www.sdss.org/dr12/data\\_access/volume/](http://www.sdss.org/dr12/data_access/volume/)
- [3] <https://indico.cern.ch/event/214784/session/6/contribution/262/material-old/paper/0.pdf>
- [4] <https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/>