Open Science Grid

Distributed Storage

Wednesday morning, 9:00am

Derek Weitzel <dweitzel@cse.unl.edu>

OSG Graduate Research Assistant

University of Nebraska – Lincoln

Who Am I?

- PhD Graduate Student from Nebraska
- Student of International Summer School for Grid Computing

Cloud Storage?

- Has anyone used cloud storage before?

Dropbox

github SOCIAL CODING

Google Drive

the simple image sharer imgur

Why did you use cloud storage?

- Why did you use Dropbox?
  - Sharing files with others?
  - Access to the files from anywhere (web)?
  - Not necessarily for storage, you still have to have a copy on your computer

What would Dropbox look like in Science?

- Would Dropbox look the same in Science?
- Do you want a copy of all your files on your computer, and some ‘Cloud’?
  - 2 copies
- Do you want it accessible from everywhere?

What would Dropbox look like in Science?

- I think Dropbox would look different
- Data is getting too big to have a ‘local’ copy on your desktop, and somewhere else.
- Want the total dataset somewhere big, and well maintained (Cloud?)
- Want only a working set on my desktop.

Outline

- Common data patterns in HTC Applications.
- Storage architecture for HTC sites
- Strategies for Managing Data.
- Reliability and the Cost of Complexity
- Prestaging data on the OSG
- Advanced Data Management Architectures

CMS Computing

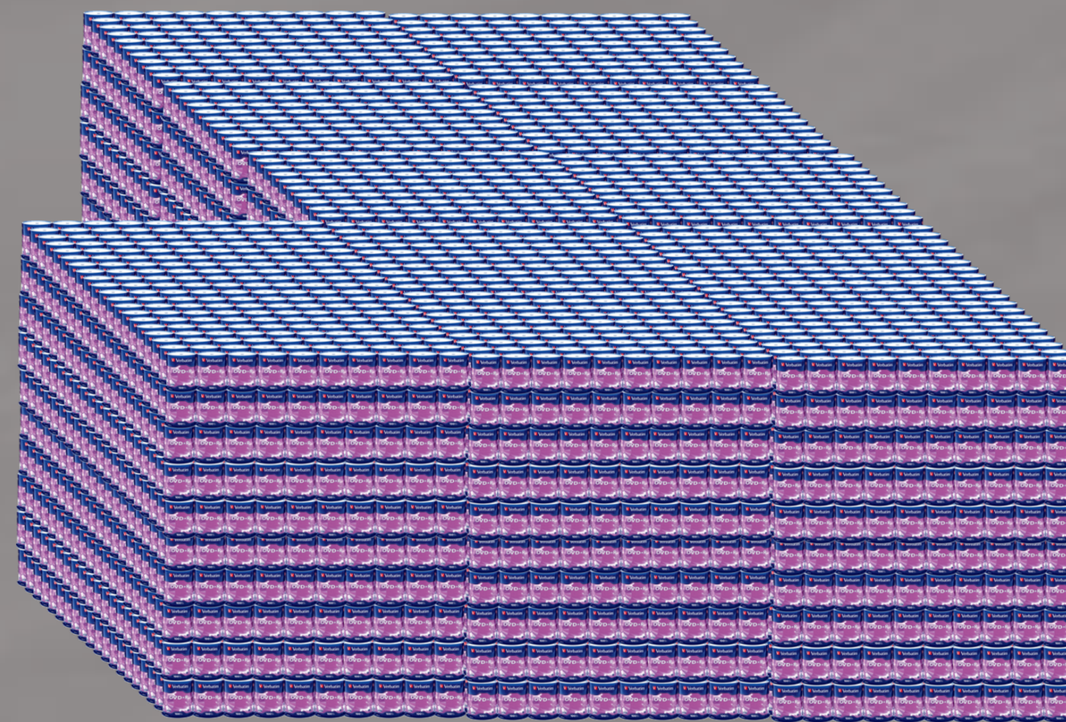
1997

E917
5 TB/run

2000

Phobos
50 TB/run

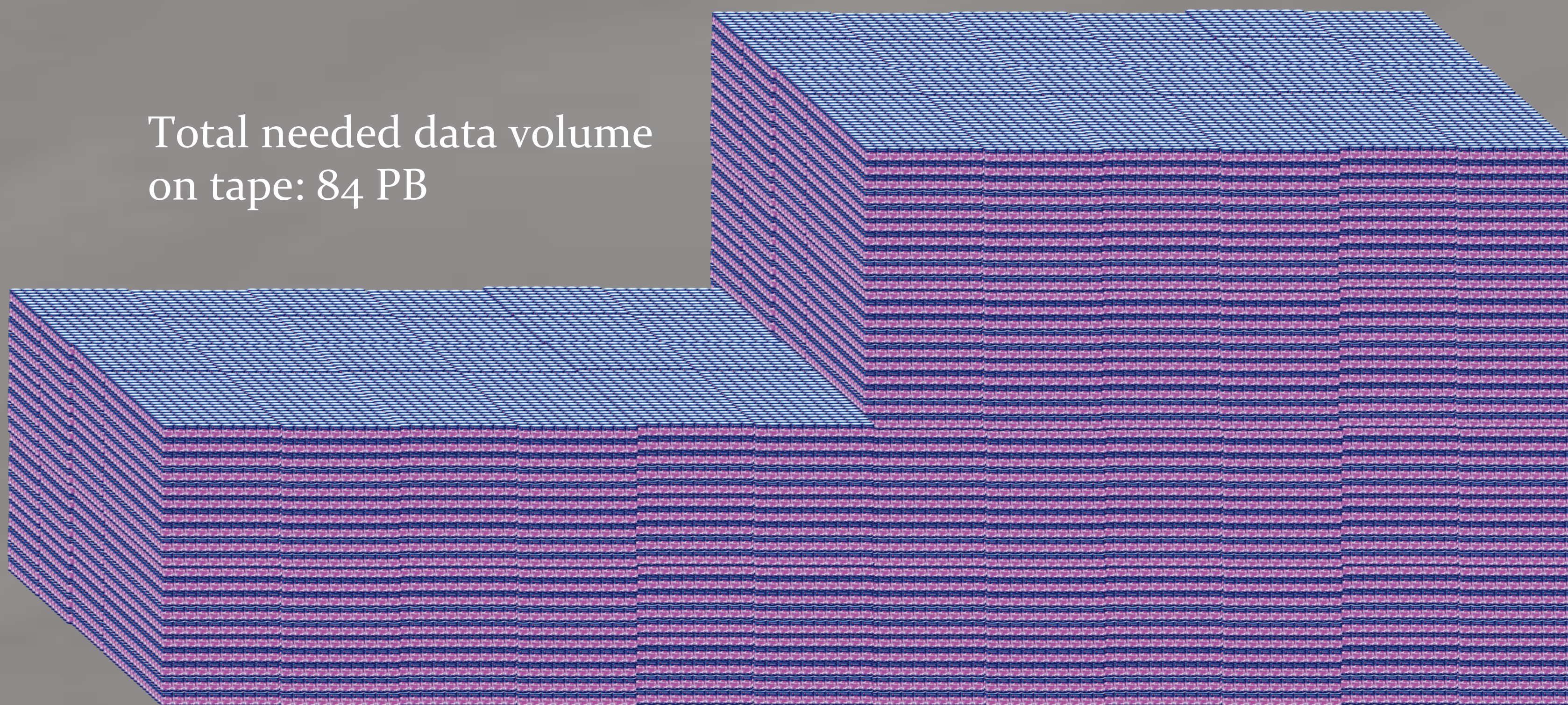
Now



CMS
6 PB raw/run

CMS Computing

Total needed data volume on tape: 84 PB



E917

Phobos

Common Storage Patterns

- There are many variations in how storage is handled.
- Not only variations from different workflows, but even within a workflow
- **Important**: Pick a proper storage strategy for you application!

Common Storage Patterns

- I'm going to highlight a few of the storage patterns seen at HTC sites

- It's important to note that moving files around is relatively easy. Error cases are the hard part of storage management.

Classifying Usage

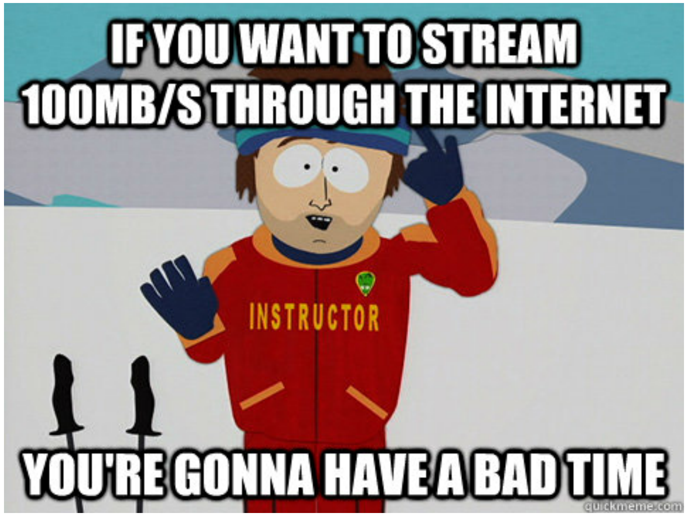
- Ask yourself, per job:
  - Events in a job's life:
    - INPUT: What should be available when the job is starting?
    - RUNTIME: What is needed while the job runs?
    - OUTPUT: What output is produced?
  - Important quantities:
    - FILES: How many files?
    - DATA: How big is the "working set" of data? How big is the sum of all the files?
    - RATES: How much data is consumed on average? At peak?

Why we care about quantities?

- FILES: How many files?
  - Many file systems cannot handle lots of small files
- DATA:
  - Working set is how much you need on a worker node for 1 ‘unit’ of work
  - How much space do you need on the file server to store the input? Output?

Why we care about quantities?

- RATES: How much data is consumed on average? At peak?
  - Even professionals have a hard time figuring this out
  - Rates determine how you should stage your data (talk about staging shortly)



Examples of usage

- Simulation
  - Small input, big out
- Searches
  - Big input, small output
- Data processing
  - ~same input and output
- Analysis
  - Big input, small output

Simulation

- Based on different input configurations, generate the physical response of a system.
  - **Input**: The HTC application must manage many input files; one per job.
  - **Runtime**: An executable reads the input and later produces output. Sometimes, temporary scratch space is necessary for intermediate results.
  - **Output**: These outputs can be small [KB] (total energy and configuration of a single molecule) or large [GB] (electronic readout of a detector for hundreds of simulated particle collisions).
    - "Huge" outputs [TB] are currently not common in HTC.

Searches

- Given a database, calculate a solution to a given problem.
  - **Input**: A database (possibly several GB) shared for all jobs, and an input file per job.
  - **Runtime**: Job reads the configuration file at startup, and accesses the database throughout the job's runtime.
  - **Output**: Typically small (order of a few MB); the results of a query/search.

Data Processing

- Transform dataset(s) into new dataset(s). The input dataset might be re-partitioned into new logical groupings, or changed into a different data format.
  - **Input**: Configuration file. Input dataset
  - **Runtime**: Configuration is read at startup; input dataset is read through, one file at a time. Scratch space used for staging output.
  - **Output**: Output dataset; similar in size to the input dataset.

Analysis

- Given some dataset, analyze and summarize its contents.
  - **Input**: Configuration file and data set.
  - **Runtime**: Configuration file is read, then the process reads through the files in the dataset, one at a time (approximately constant rate). Intermediate output written to scratch area.
  - **Output**: Summary of dataset; smaller than input dataset.

Open Science Grid

OSG Anti-Patterns

- Just as we want to identify common successful patterns, we want to identify common patterns that are unsuccessful on OSG.
  - Files larger than >5GB.
  - Requiring a (POSIX) shared file system.
  - Lots of small files (more than 1 file per minute of computation time).
  - Jobs consuming more than 10GB per hour, or needing scratch space more than 5GB.
  - Locking, appending, or updating files.
- When using OSG resources opportunistically, the effective limitations may be more restrictive than above!

Exercise to help identify problems

- Remember, we want to identify:
  - Input data – How much data is needed for the entire workflow?
  - Working set – How much data is needed to run 1 unit of work? Including possible temporary files.
  - Output data – How much data is output to the workflow.

Open Science Grid

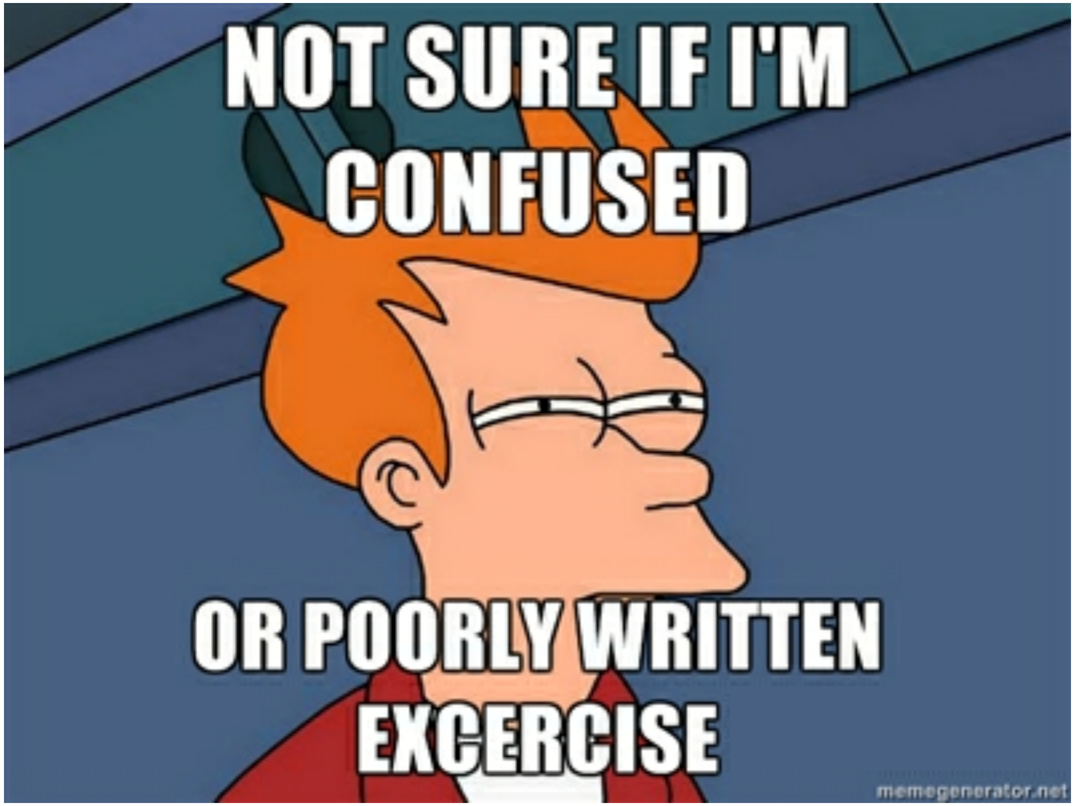
Questions?

- Questions? Comments?
  - Feel free to ask me questions:

    Derek Weitzel <dweitzel@cse.unl.edu>
- Upcoming sessions
  - 9:30-10:15
    - Hands-on exercises
  - 10:15 – 10:30
    - Break
  - 10:30 – 12:00
    - More!

Exercise

- https://opensciencegrid.org/bin/view/Education/OSGSS2013StorageEx1



Lecture 2: Using Remote Storage Systems

Derek Weitzel <dweitzel@cse.unl.edu>

Last Exercise

- What where some of the numbers for the last exercise?

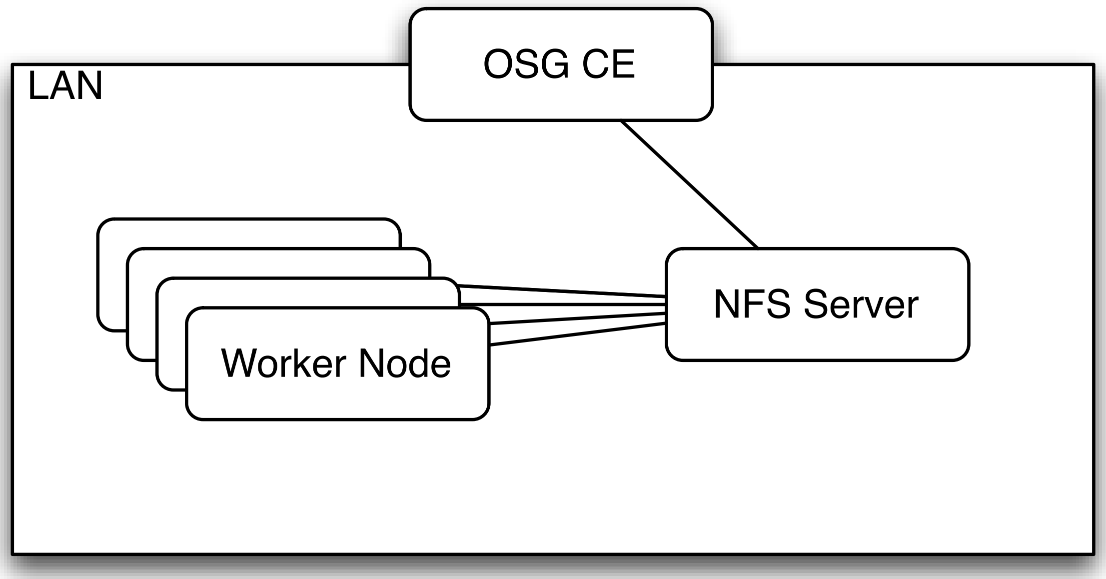
Last Exercise

- Yeast:
  - Data: 22MB (executable = 15MB, why?)
  - Rate
    - MB/s = 22MB / .3s = 73MB/s
    - Files = 11 / .3s = 36/s
- Compare:
  - Spinning disk = ~60MB/s

Storage Architectures on the OSG

- Lets look at how storage is structured on the OSG
- This will give you an idea of where the bottlenecks can be.
- Also can help when forming data solutions on your own.

Storage at OSG CEs



- All OSG sites have some kind of shared, POSIX-mounted storage (typically NFS).* This is almost never a distributed or high-performance file system
- This is mounted and writable on the CE.*
- This is readable (sometimes read-only) from the OSG worker nodes.

*Exceptions apply! Sites ultimately decide

Why Not?

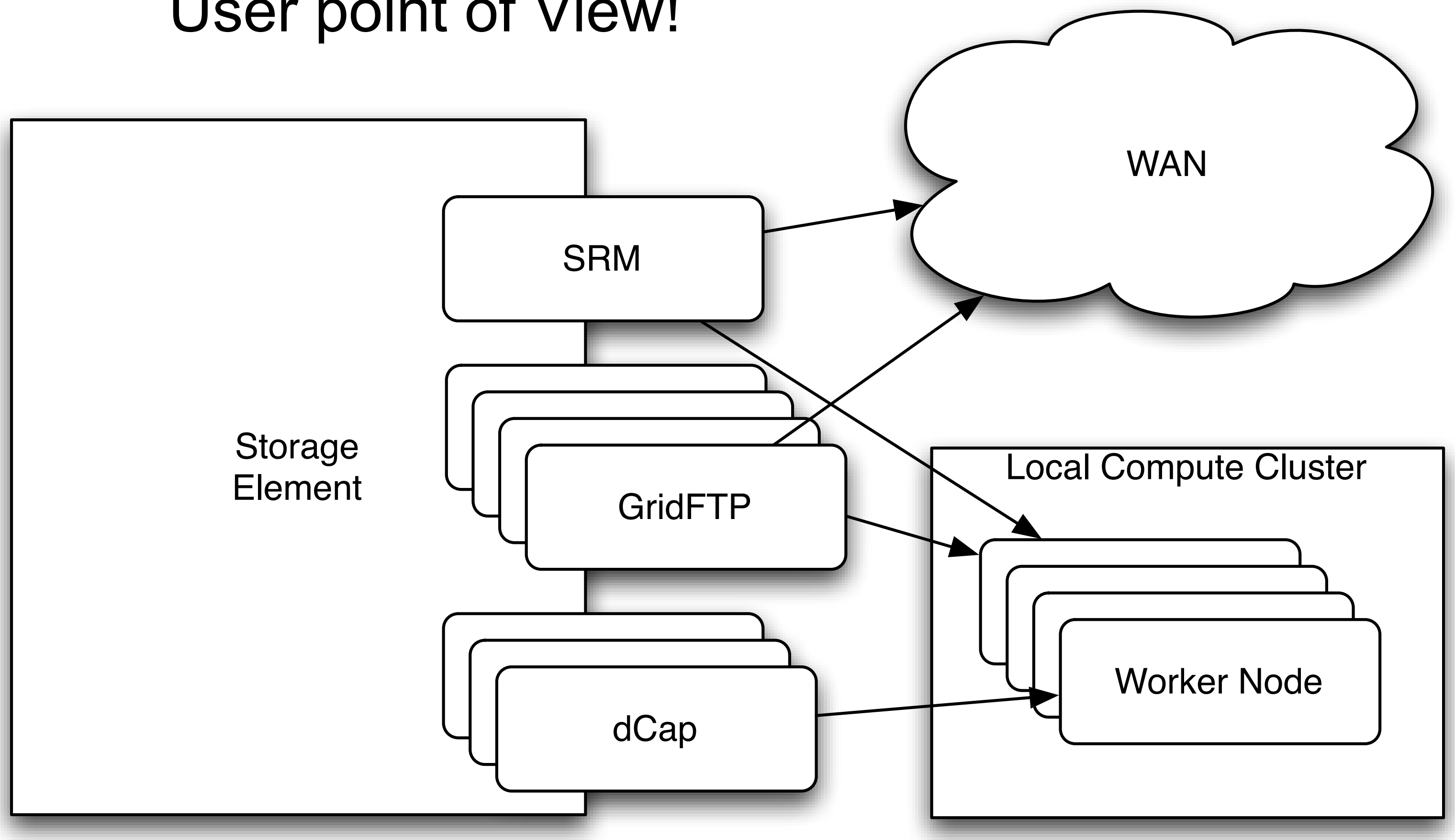
- This setup is called the "classic SE" setup, because this is how the grid worked circa 2003.
  - ***Why didn't this work*?**
- High-performance filesystems not reliable or cheap enough.
- Scalability issues.
- Difficult to manage space.

Storage Elements

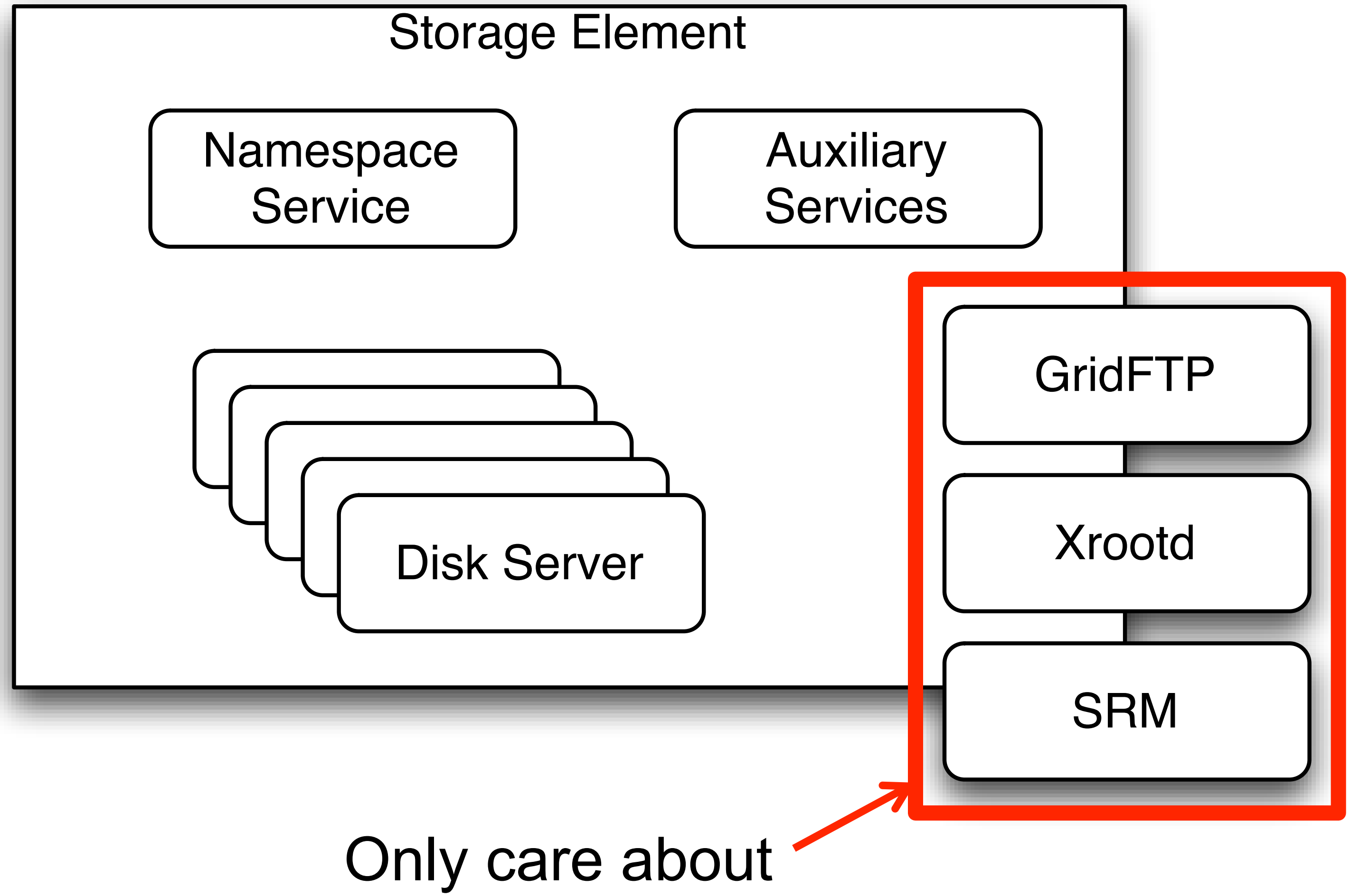
- In order to make storage and transfers scalable, sites set up a separate system for storage (the *Storage Element*).
- Most sites have an attached SE, but there's a wide range of scalability.
- These are separated from the compute cluster; normally, you interact it via a get or put of the file.
  - Not POSIX!

Storage Elements on the OSG

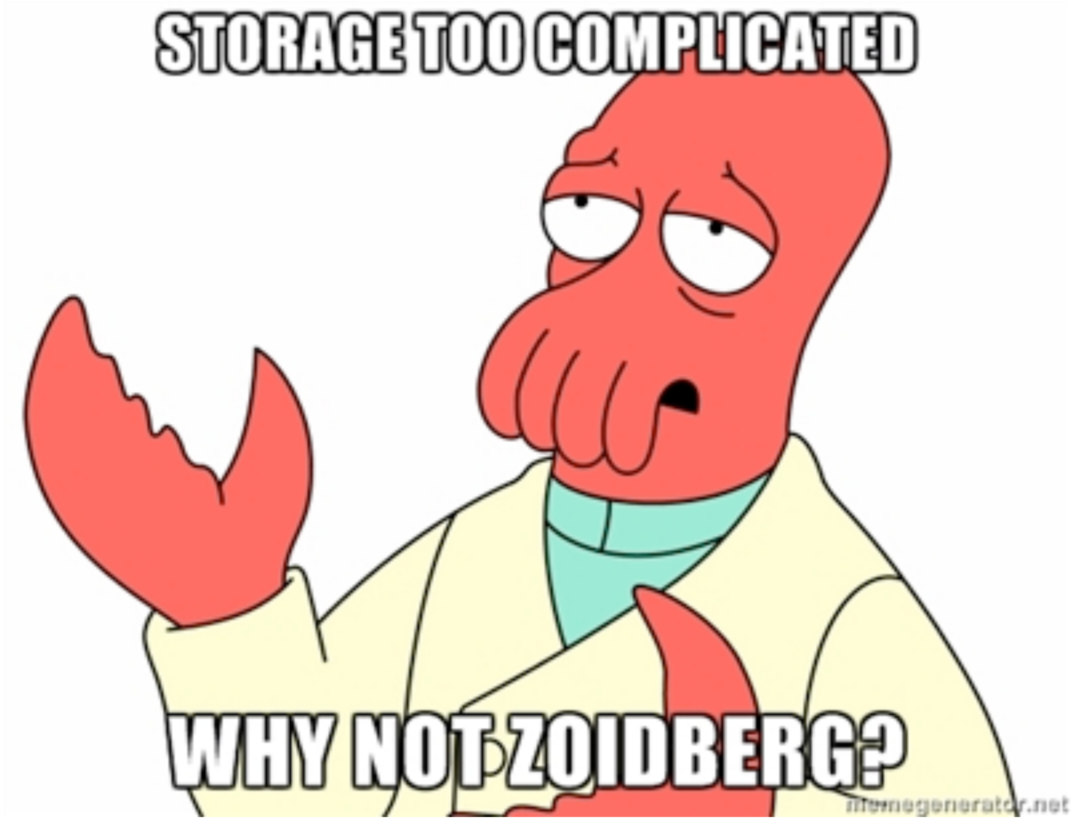
User point of View!



SE Internals



Complicated?



GridFTP in One Slide

- An set of extensions to the classic FTP protocol.
- Two most important extensions:
  - **Security**: Authentication, encryption, and integrity provided by GSI/X509. Use proxies instead of username/password.
  - **Improved Scalability**: Instead of transferring a file over one TCP connection, multiple TCP connections may be used.

SRM

- SRM = Storage Resource Management
- SRM is a web-services-based protocol for doing:
  - Metadata Operations.
  - Load-balancing.
  - Space management.
- This allows us to access storage remotely, and to treat multiple storage implementations in a homogeneous manner.

Data Access Methods

- To go with our archetypical problems, we'll have a few common approaches to implementing solutions:
  - Job Sandbox
  - Prestaging
  - Caching
  - Remote I/O
- A realistic solution will combine multiple methods.
- Descriptions to follow discuss the common case; there are always exceptions or alternates.

Job Sandbox

- Data sent with jobs.
- The user generates the files on the submit machines.
- These files are copied from submit node to worker node.
  - With Condor, a job won't even start if the sandbox can't be copied first. You can always assume it is found locally.
- What are the drawbacks of the job sandbox?

Prestaging

- Data placed into some place “near” to where the jobs are run.
- By increasing the number of locations of the data and doing intelligent placement, we increase scalability.
  - In some cases, prestaging = making a copy at each site where the job is run.
  - Not always true; a single storage element may be able to support the load of many sites.

Caching

- A cache transparently stores data close to the user process. Future requests can be served from the cache.
  - User must place file/object at a source site responsible for keeping the file permanently. Cache must be able to access the file at its source.
  - Requesting a file will bring it into the cache.
  - Once the cache becomes full, an eviction policy is invoked to decide what files to remove.
  - The cache, not the user, decides what files will be kept in the cache.

Remote I/O

- Application data is streamed from a remote site upon demand. I/O calls are transformed into network transfers.
  - Typically, only the requested bytes are moved.
  - Often, data is streamed from a common source site.
  - Can be done transparently so the user application doesn't know the I/O isn't local.
- Can be effectively combined with caching to scale better.

Scalability

- For each of the four previous methods (sandboxes, prestaging, caching, remote I/O), how well do they scale?
  - What are the limitations on data size?
  - What resources are consumed per user-process?
  - What are the scalability bottlenecks?

The Cost of Reliability

- For each of the four previous methods, what must be done to create a reliable system?
  - What's the cost for tracking the location of files?
  - What recovery must be done on failures?
    - What must be done manually, or require extra infrastructure?
  - What is the most critical bottlenecks or points of failure?
    - How do these tie into the job submission system?

Comparing Compute and Storage

- The "shared resource" for computing is the gatekeeper.
  - One badly behaving user can overload the gatekeeper, preventing new jobs from starting.
  - However, once jobs are started, they are mostly independent.
    - There are some shared aspects: what are they? Why am I not concerned about them?

Comparing Compute and Storage

- Storage is different:
  - It is often used throughout the job's lifetime, especially for remote I/O.
  - One badly behaved user can crash the storage resource – or at least severely degrade.
  - Opportunistic usage of storage via prestaging does not automatically have a limited lifetime.
    - Most users assume that data, once written to a SE, will be retrievable.

Prestaging Data on the OSG

- Prestaging is currently the most popular data management method on the OSG, but requires the most work.
  - Requires a **transfer system** to move a list of files between two sites. Example systems: Globus Online, Stork, FTS.
  - Requires a **bookkeeping system** to record the location of datasets. Examples: LFC, DBS.
  - Requires a mechanism to **verify** the current validity of file locations in the bookkeeping systems. Most systems are ad-hoc or manual.

Previous example?

- In the first exercise, you formulated a data plan for BLAST.
- Now, you are going to implement the data movement of blast using pre-staging.

Exercise Time!

- https://twiki.grid.iu.edu/bin/view/Education/OSGSS2013StorageEx2

Open Science Grid

Lecture 3: Caching and Remote I/O

Derek Weitzel <dweitzel@cse.unl.edu>

HTTP Caching is easy on the OSG

- Widely deployed caching servers (required by other experiments)

- Tools are easy to use
  - curl, wget

- Servers are easy to setup
  - Just need http server, somewhere

Open Science Grid

HTTP Caching

- In practice, you can cache almost any URL
- Site caches are typically configured to cache any URL, but only accept requests from WN's
- Sites control the size and policy for caches, keep this in mind.

HTTP Caching - Pitfalls

- `curl` – Have to add special argument to use caching:
  - `curl -H "Pragma:” <url>`

- Services like Dropbox and Google Docs explicitly disable caching
  - Dropbox: `cache-control: max-age=0`
  - Google Docs: `Cache-Control: private, max-age=0`
  - They also use uncachable HTTPS

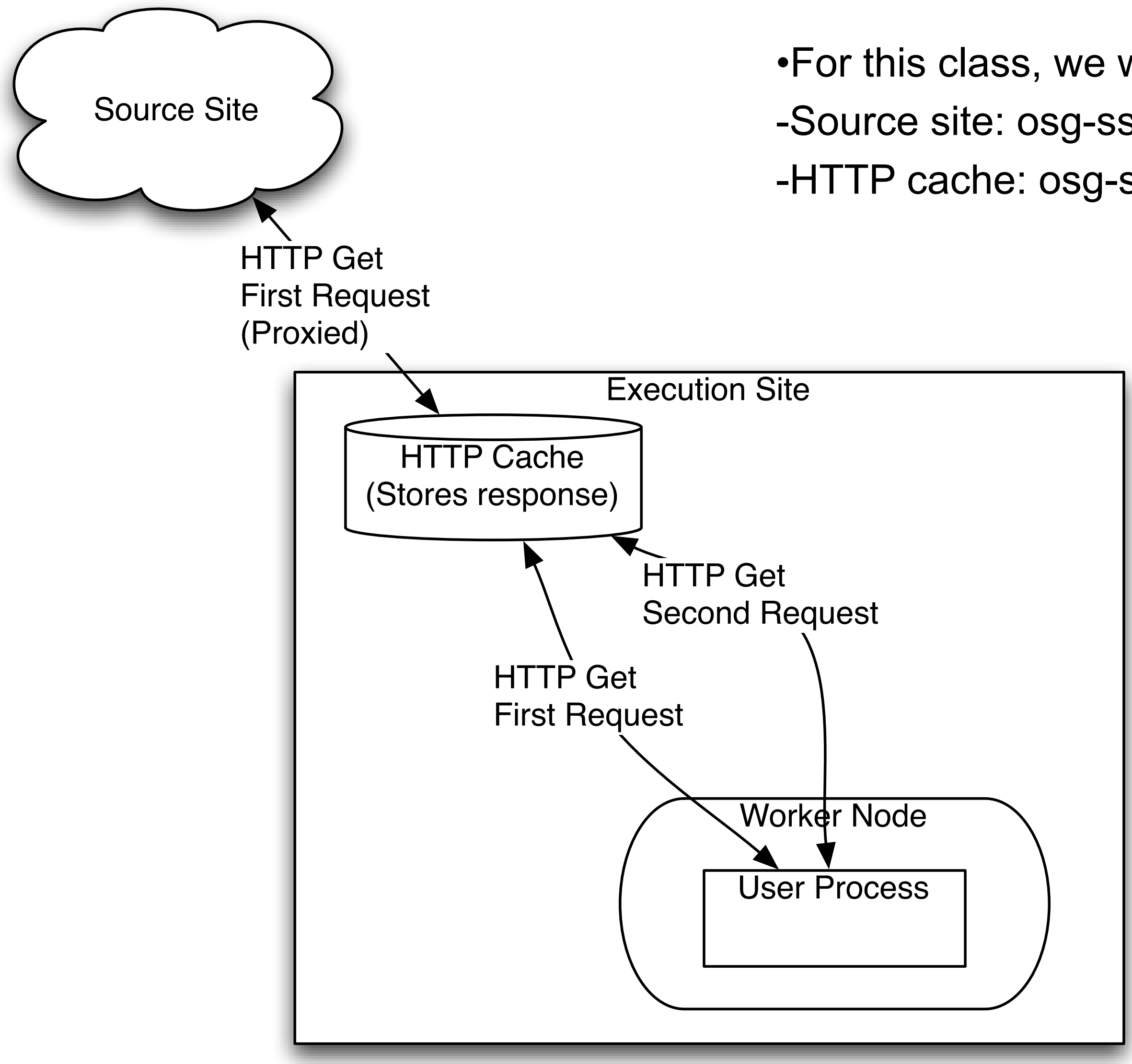
HTTP Caching

- The HTTP protocol is perhaps the most popular application-layer protocol on the planet.
- Built-in to HTTP 1.1 are elaborate mechanisms to use HTTP through a proxy and for caching.
- There are mature, widely-used command-line clients for the HTTP protocol on Linux.
  - Curl and wget are the most popular; we will use wget for the exercises.
    - Oddly enough, curl disables caching by default.

HTTP Caching: What do you need?

- All you need is a public webserver… somewhere.
- Will pull down files to the worker node through squid cache.

HTTP Cache Dataflow



- For this class, we will use:
- -Source site: osg-ss-submit.chtc.wisc.edu
- -HTTP cache: osg-ss-se.chtc.wisc.edu

Proxy Request

- Here's the HTTP headers for the request that goes to the proxy osg-ss-se.cs.wisc.edu:

```
GET http://osg-ss-submit.chtc.wisc.edu/~dweitzel/stuff.html HTTP/
1.0
User-Agent: Wget/1.11.4 Red Hat modified
Accept: */*
Host: osg-ss-submit.chtc.wisc.edu
```

Proxy Request

- Response:

```
HTTP/1.0 200 OK
Date: Tue, 19 Jun 2012 22:53:26 GMT
Server: Apache/2.2.3 (Scientific Linux)
Last-Modified: Tue, 19 Jun 2012 21:26:56 GMT
ETag: "136d0f-59-4c2d9f120e800"
Accept-Ranges: bytes
Content-Length: 89
Content-Type: text/html; charset=UTF-8
X-Cache: HIT from osg-ss-se.chtc.wisc.edu
X-Cache-Lookup: HIT from osg-ss-se.chtc.wisc.edu:3128
Via: 1.0 osg-ss-se.chtc.wisc.edu:3128 (squid/2.6.STABLE21)
Proxy-Connection: close
```

HTTP Proxy

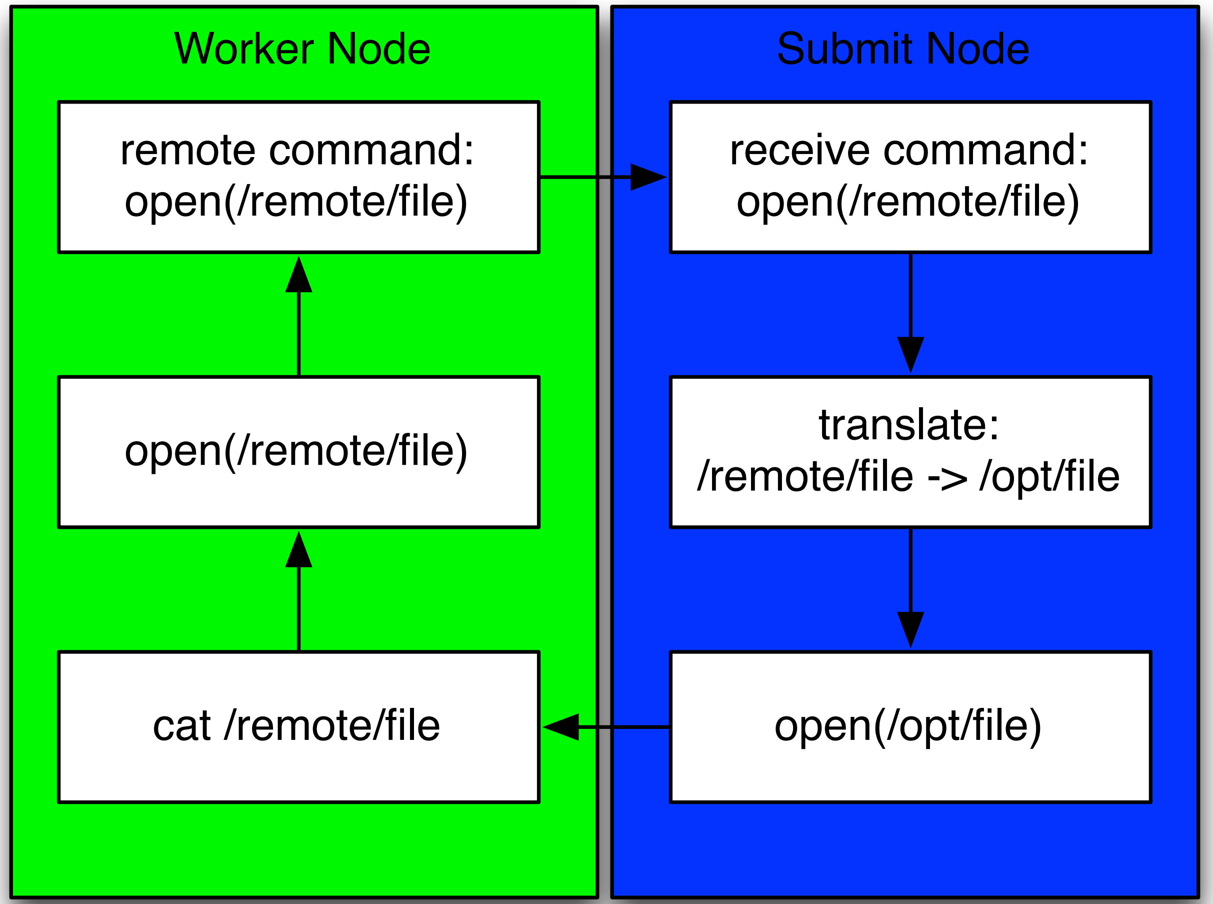
- HTTP Proxy is very popular for small VO’s
- Easy to manage, easy to implement
- Highly recommend HTTP Proxy for files less than 100MB.

Now on to Remote I/O

- Do you only read parts of the data (skip around in the file?)
- Do you have data that is only available on the submit host?
- Do you have an application that requires a specific directory structure?

Remote I/O is for you

Remote I/O



Worker Node
remote command: open(/remote/file)
open(/remote/file)
cat /remote/file
Submit Node
receive command: open(/remote/file)
translate: /remote/file -> /opt/file
open(/opt/file)

Open Science Grid

Remote I/O

The Cooperative Computing Lab

- We will perform Remote I/O with the help of Parrot from ND.
- It redirects I/O requests back to the submit host.
- Can ‘pretend’ that a directory is locally accessible.

Examples of Remote I/O

- What does Remote IO look like in practice?

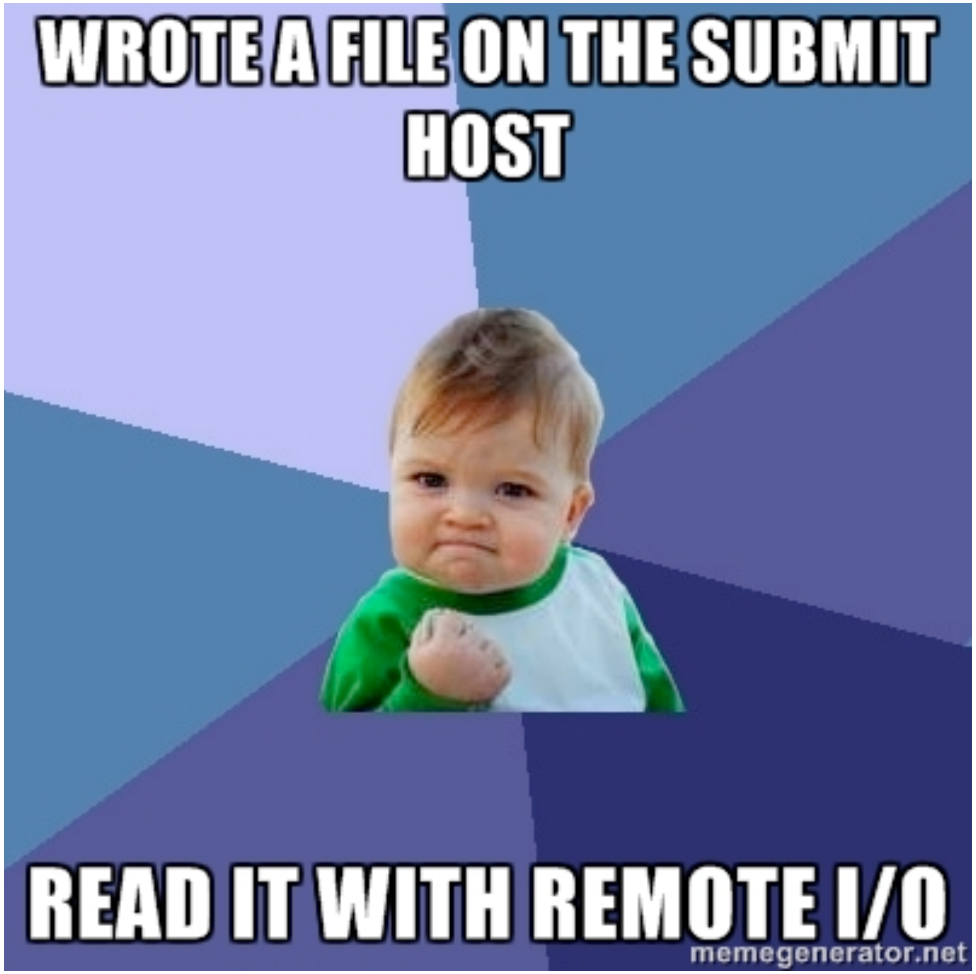
submit host:

```
$ echo "hello from submit host" > newfile
```

script on the worker node (in Nebraska?)

```
$ cat /chirp/CONDOR/home/dweitzel/newfile
hello from submit host
$
```

Celebrate!


WROTE A FILE ON THE SUBMIT HOST
READ IT WITH REMOTE I/O
memegenerator.net

Downsides of Remote I/O

- What are some downsides to Remote I/O?
- All I/O goes through the submit host.
- Distributed remote I/O is hard (though can be done, see XrootD)

Now lets do it!

- https://twiki.grid.iu.edu/bin/view/Education/OSGSS2013StorageEx3