

Data Management for HTC

Tuesday Morning

Brian Bockelman,

bbockelm@cse.unl.edu

University of Nebraska-Lincoln

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

Common data patterns in HTC Applications

- I'm going to give a few common patterns for accessing data observed at HTC-centric sites.
- This is not meant to be exhaustive; for each “theme”, there's a huge number of variations.
 - Not only variations of a single pattern, but several patterns in a workflow.

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?

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.

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!

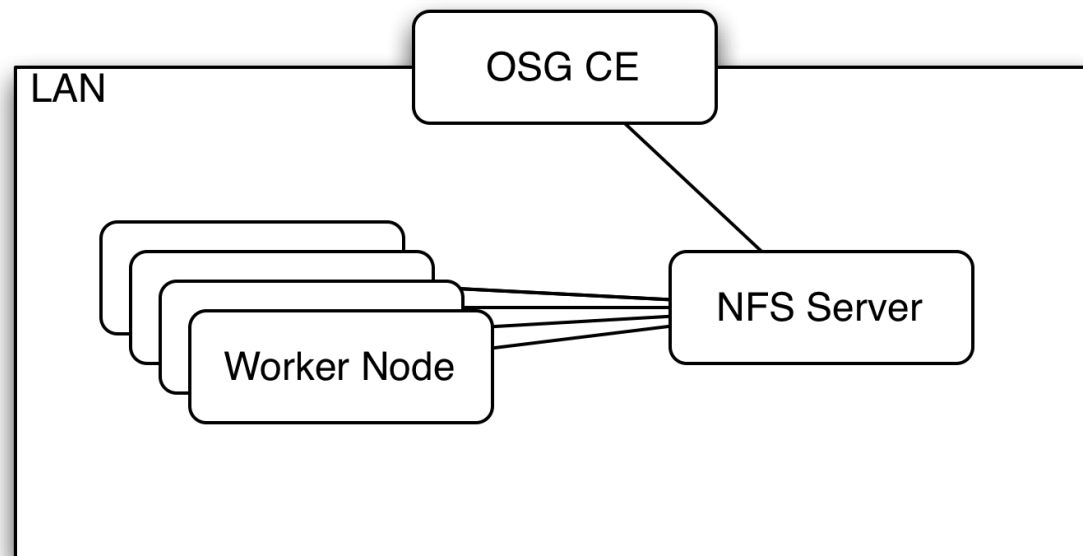
Break for Exercise #1

- For each scenario in the handout, determine what pattern it is closest to.
- Identify the input / runtime needs / output of the job.
- Is the job appropriate to run on the OSG?
- If there's not enough information to determine this, ask!
 - In real life, the researcher asks for help with incomplete information: you must learn what questions to ask.

Storage Architectures on the OSG

- Given the previous problems, I want to discuss common solution patterns.
 - Before that, we need a short primer on how storage is architected on the OSG.

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?

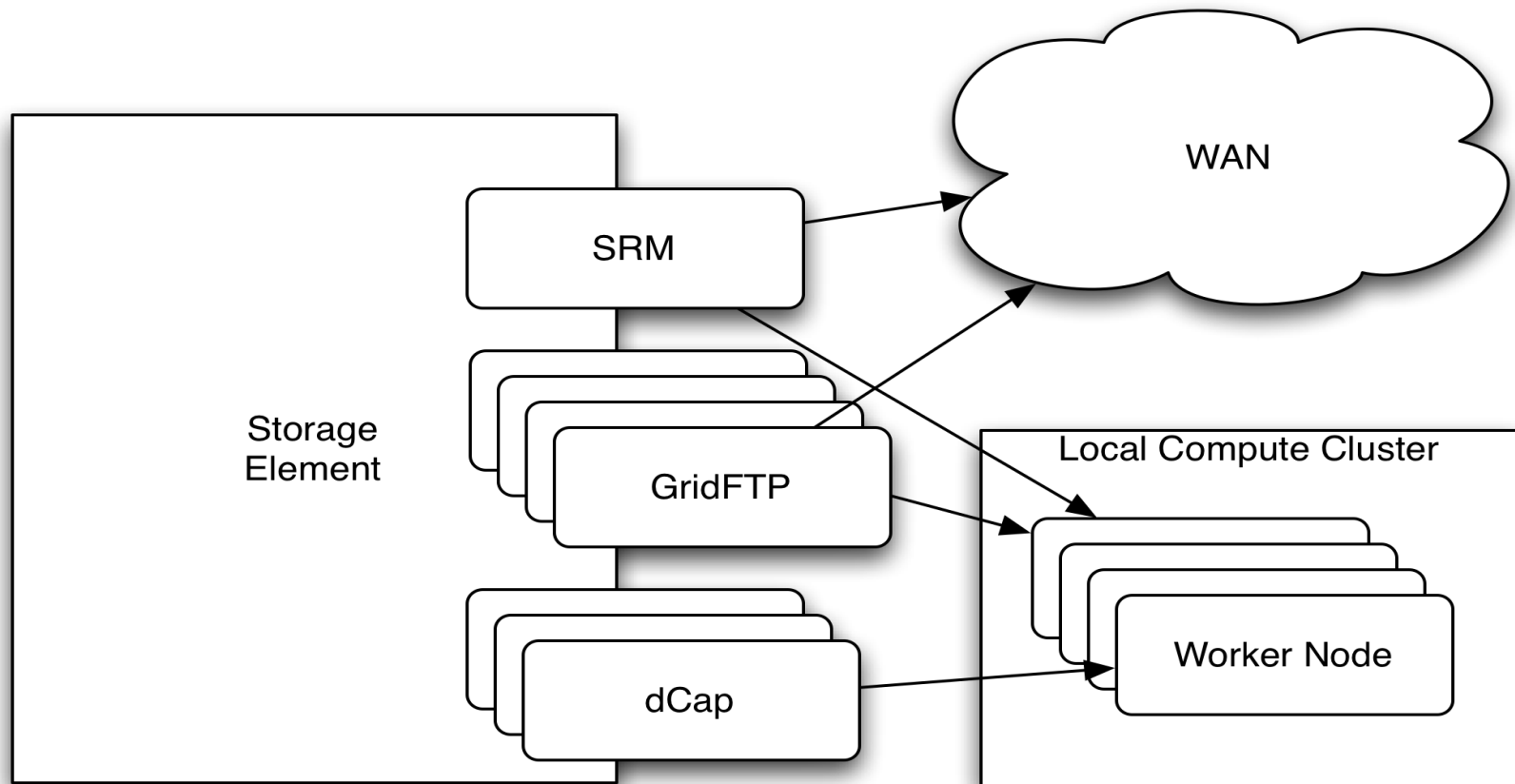
- This setup is called the “classic SE” setup, because this is how the grid worked circa 2003.
 - *Why didn't this work?*
- High-performance filesystems not reliable or cheap enough.
- Scalability issues.
- Difficult to manage space.

Storage Elements

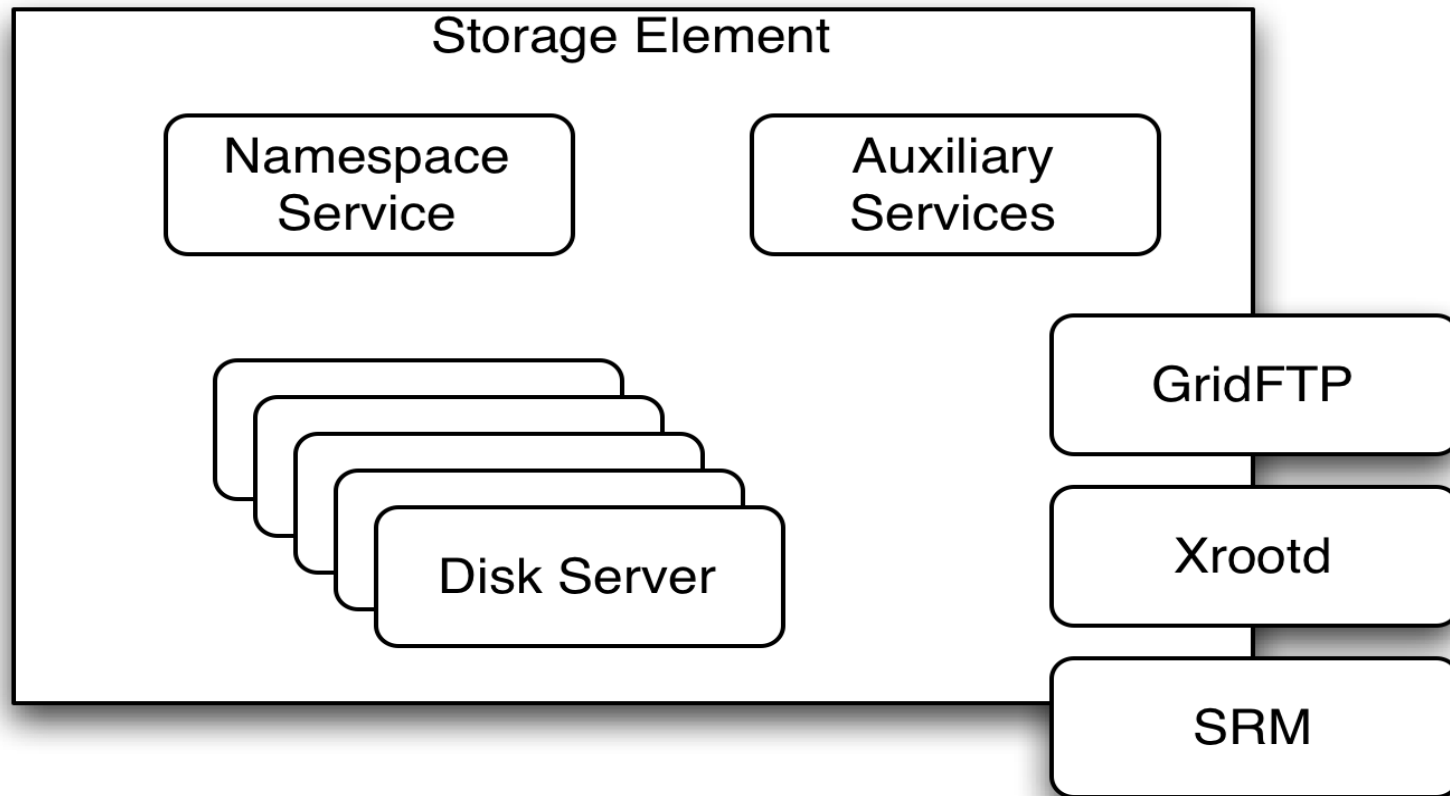
- In order to make storage and transfers scalable, sites set up a separate system for storage (the *Storage Element*).
- Most sites have an attached SE, but there's a wide range of scalability.
- These are separated from the compute cluster; normally, you interact it via a get or put of the file.
 -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

Storage Elements on the OSG

User point of View!



SE Internals



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.

Break for Exercise #1

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.

Prestaging Data on the OSG

- In the exercises, we'll do some basic prestaging:
 - You will write a script to do the file movement.
 - Destinations and sources will be specified manually.
 - No bookkeeping will be done. Once written, we will assume files are always available and valid.
 - We will use the SRM protocol to handle metadata operations and load-balancing, and GridFTP for transfers.

Break for Exercise #2

- <https://twiki.grid.iu.edu/bin/view/Education/OSGSS2011StorageEx2>
- Note: To complete this exercise, you must fill your user name into the form and hit “Customize”.

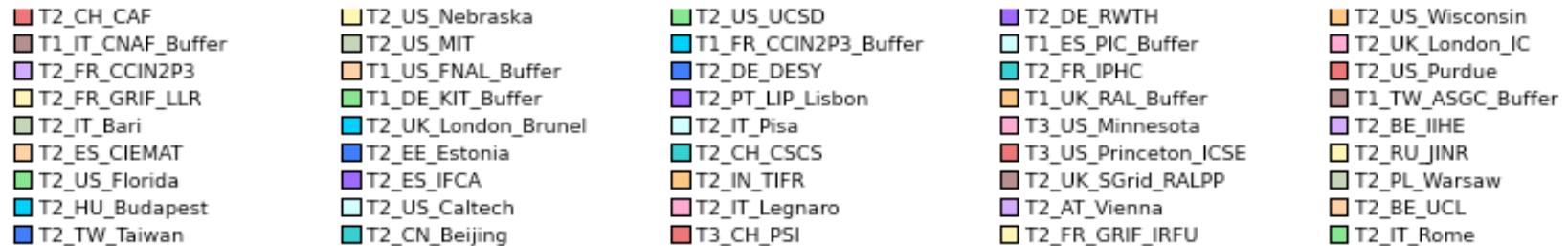
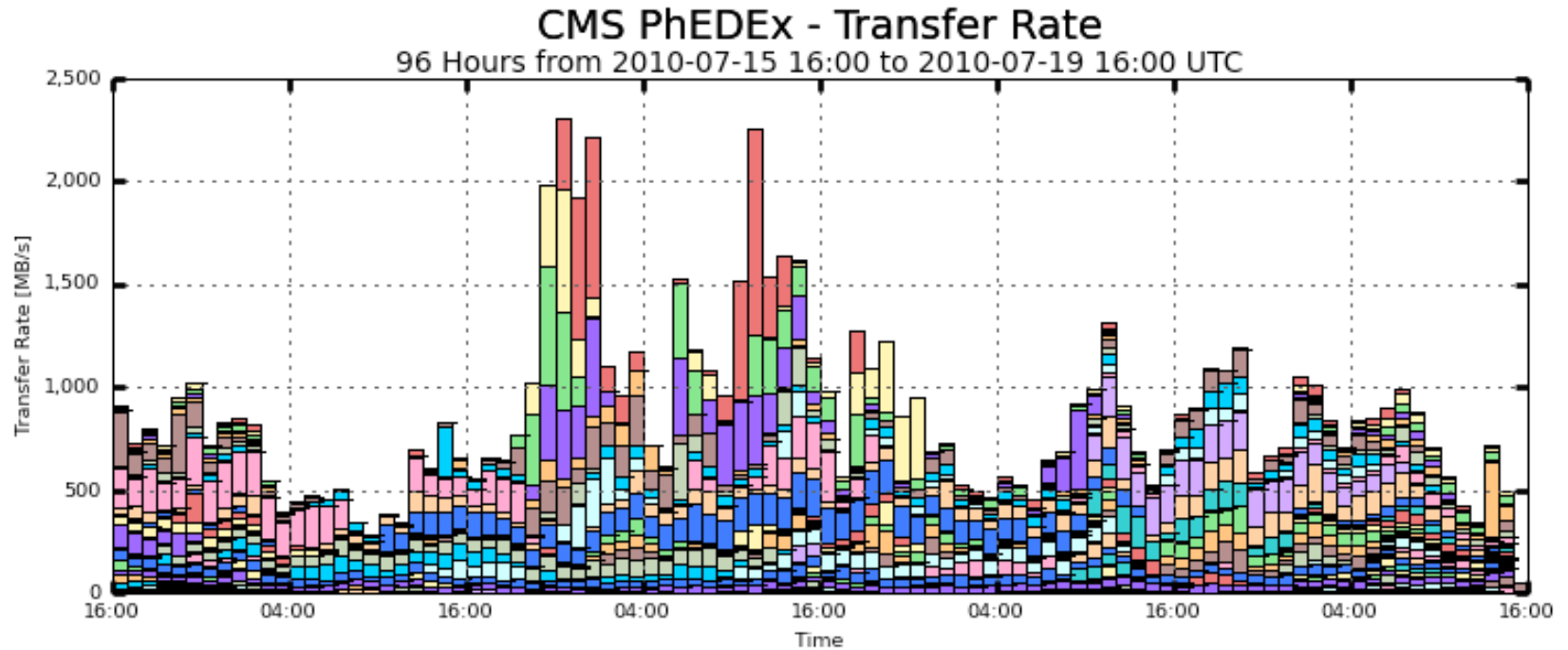
Example Solution: PhEDEx

- PhEDEx is the CMS data transfer system.
- Transfers are done using FTS.
- Bookkeeping is done via a custom Oracle system.
- Destination sites are subscribed to datasets by users.
- Source sites are selected per-file by intelligent routing mechanisms.

PhEDEx

- Each site runs a set of processes called Agents to manage the download activity.
 - Requires admin time at each participating site.
- Full-featured monitoring via the PhEDEx website.
- Resulting system is highly effective, extremely scalable, but probably requires a large amount of effort.

PhEDEx Snapshot



Maximum: 2,307 MB/s, Minimum: 54.07 MB/s, Average: 872.36 MB/s, Current: 54.07 MB/s

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 Cache Dataflow



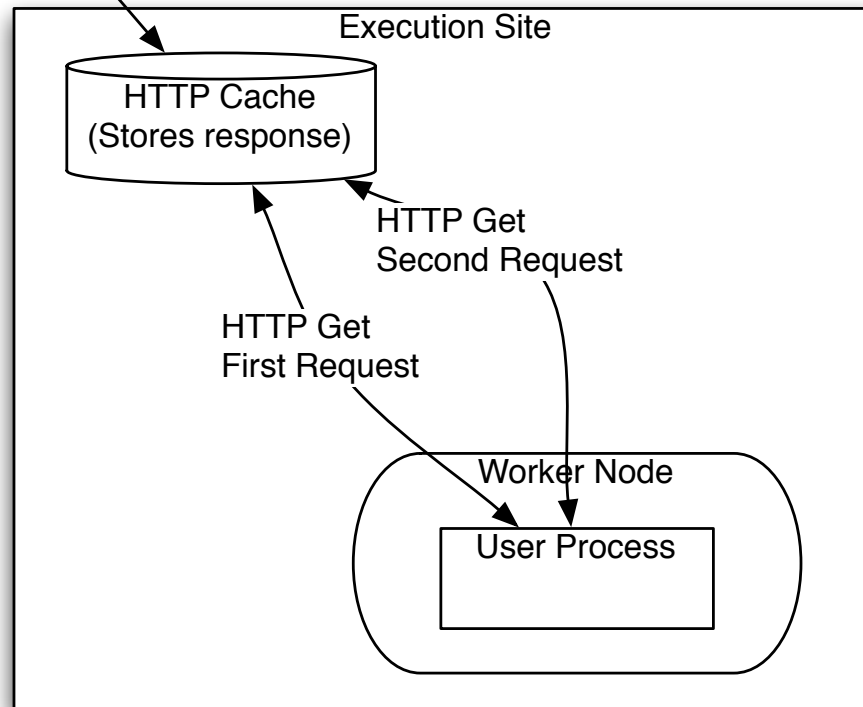
Source Site

HTTP Get
First Request
(Proxied)

For this class, we will use:

-Source site: vdt-itb.cs.wisc.edu

-HTTP cache: osg-edu-se.cs.wisc.edu



Proxy Request

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

GET http://vdt-itb.cs.wisc.edu/dm_exercises_part3/bbockelm/yeast.aa.psq HTTP/1.0
User-Agent: Wget/1.11.4 Red Hat modified
Accept: */*
Host: vdt-itb.cs.wisc.edu

Proxy Response

- Here is the response headers from osg-edu-se.cs.wisc.edu:

HTTP/1.0 200 **OK**

Date: Fri, 24 Jun 2011 22:43:53 GMT

Server: Apache/2.2.3 (Scientific Linux)

Last-Modified: Fri, 24 Jun 2011 22:43:10 GMT

ETag: "2f89a4-2d79f1-4a67ced0c3b80"

Accept-Ranges: bytes

Content-Length: 2980337

Content-Type: text/plain; charset=UTF-8

Age: 7

X-Cache: HIT from osg-edu-se.cs.wisc.edu

Via: 1.0 osg-edu-se.cs.wisc.edu:3128 (squid/2.6.STABLE23)

Proxy-Connection: close

Break for Exercise #3

- <https://twiki.grid.iu.edu/bin/view/Education/OSGSS2011StorageEx3>
- Note: To complete this exercise, you must fill your user name into the form and hit “Customize”.