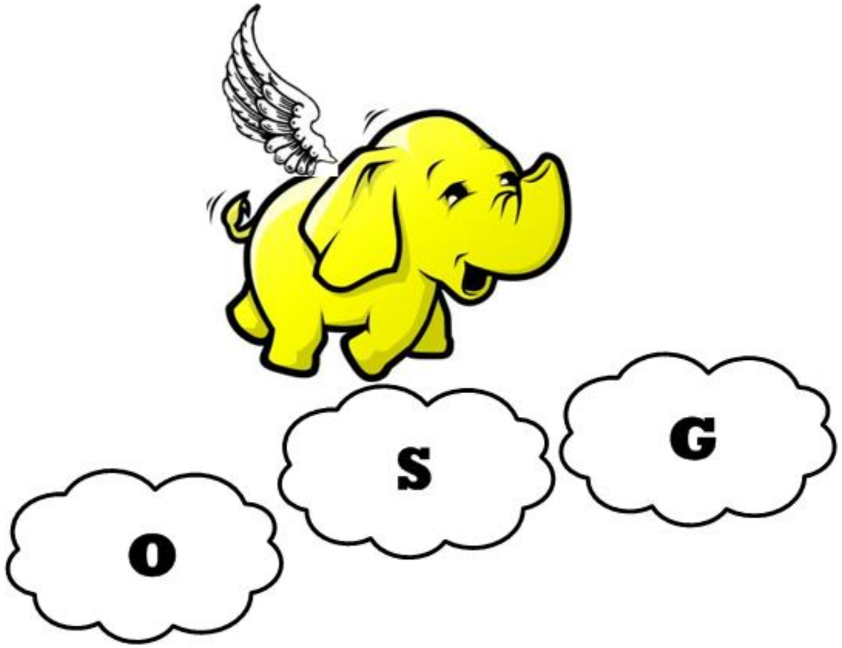


HOG: Hadoop on the Grid

Derek Weitzel

University of Nebraska – Lincoln

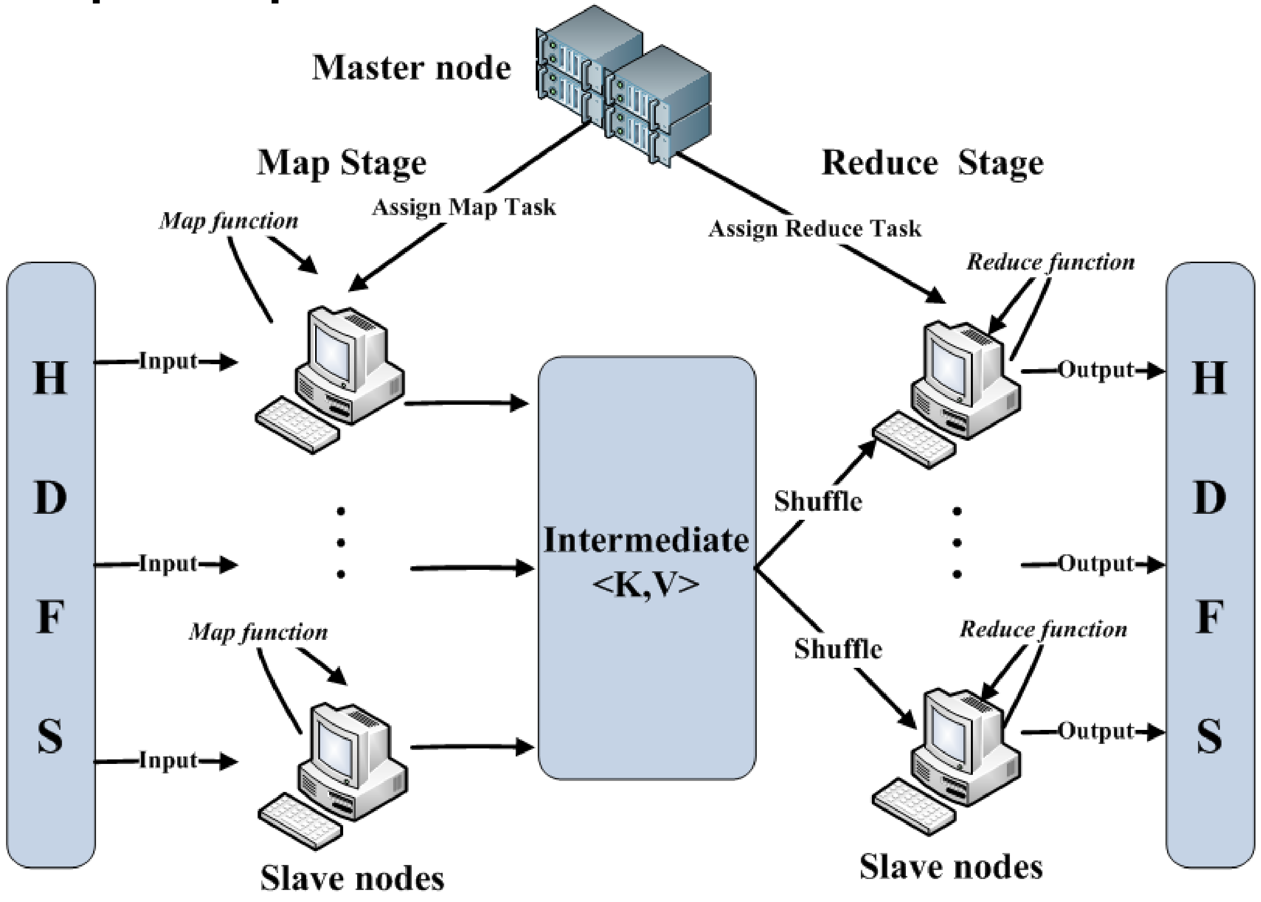
Outline

Introduction

- Combining 2 components:
  - **Hadoop**: Distributed, fault tolerant data intensive computing framework.
  - **Open Science Grid** (OSG): Nationally distributed computing centers linked that opportunistically share resources.

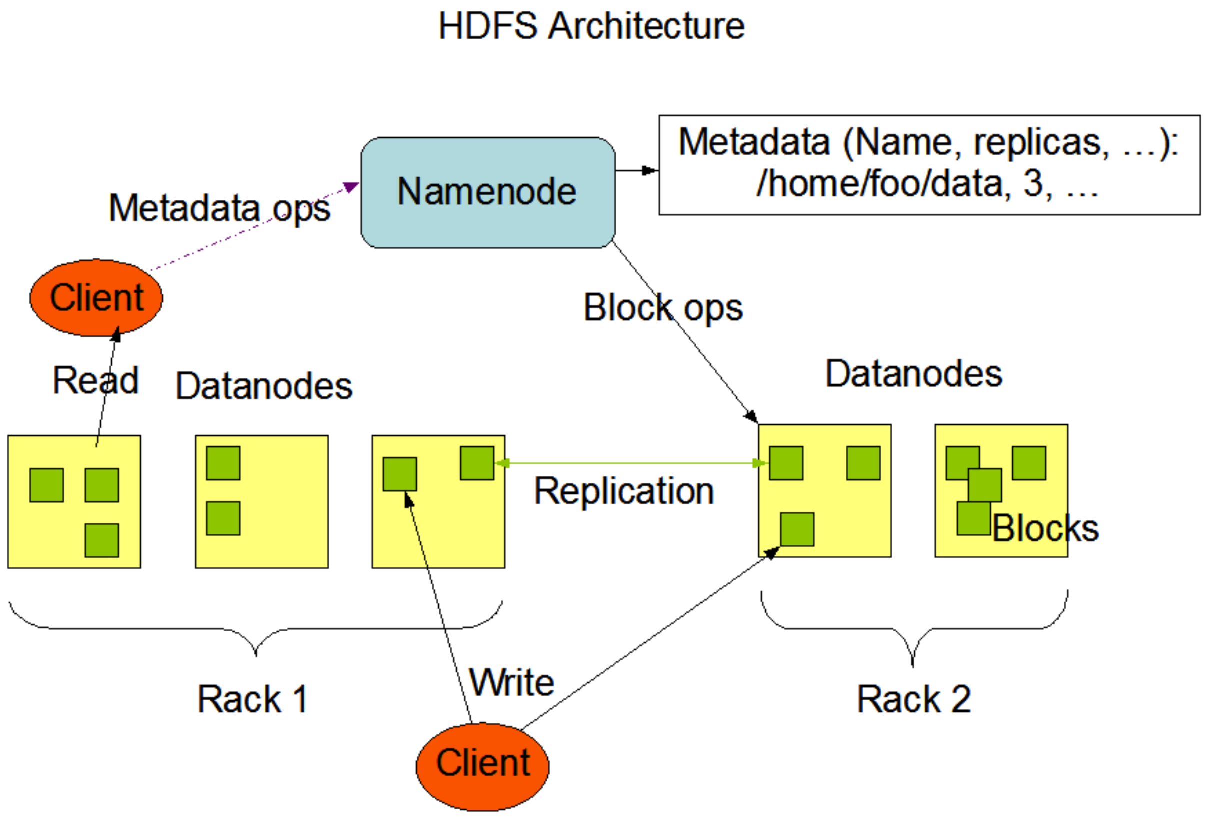
Introduction

- Hadoop MapReduce



Introduction

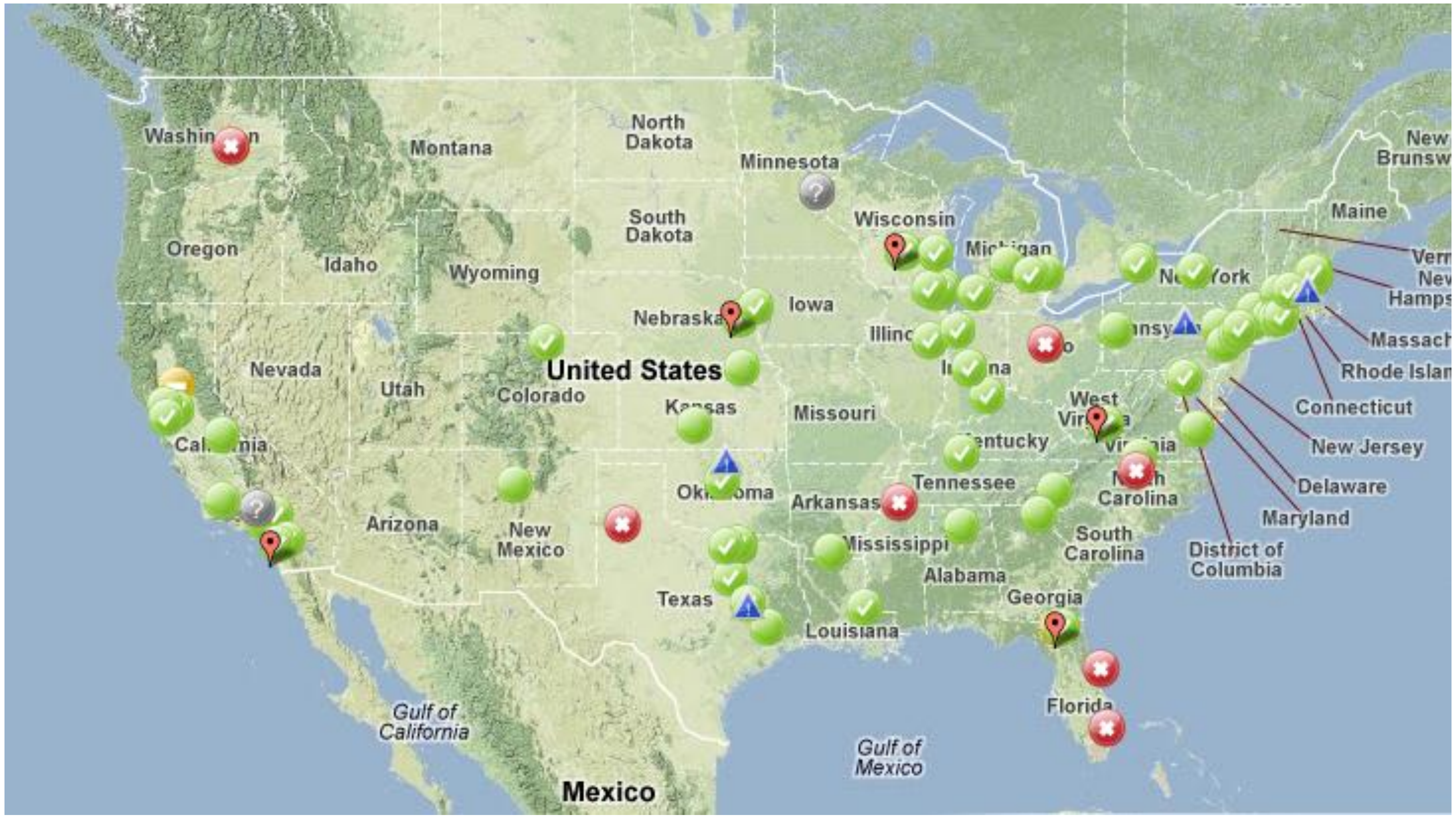
- Hadoop HDFS



Picture adopted from www.hadoop.com

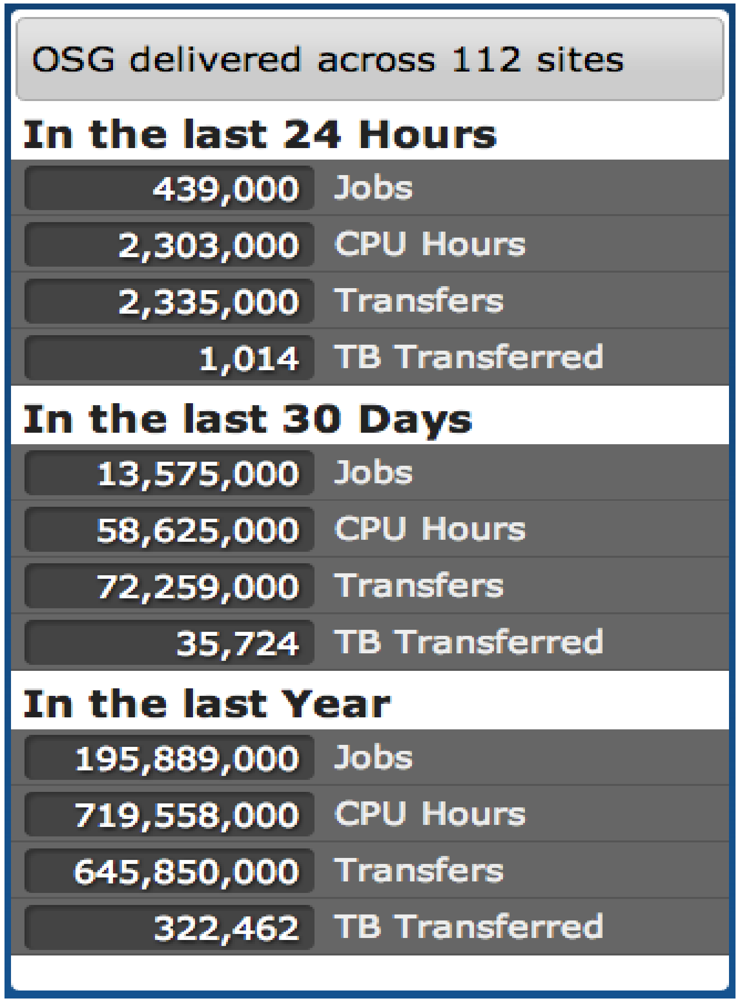
Introduction

- Open Science Grid – 112 Sites



Introduction

- OSG is large national grid
- Used by researchers at universities and labs.
- Primarily a Data Intensive Grid



OSG delivered across 112 sites

In the last 24 Hours

439,000	Jobs
2,303,000	CPU Hours
2,335,000	Transfers
1,014	TB Transferred

In the last 30 Days

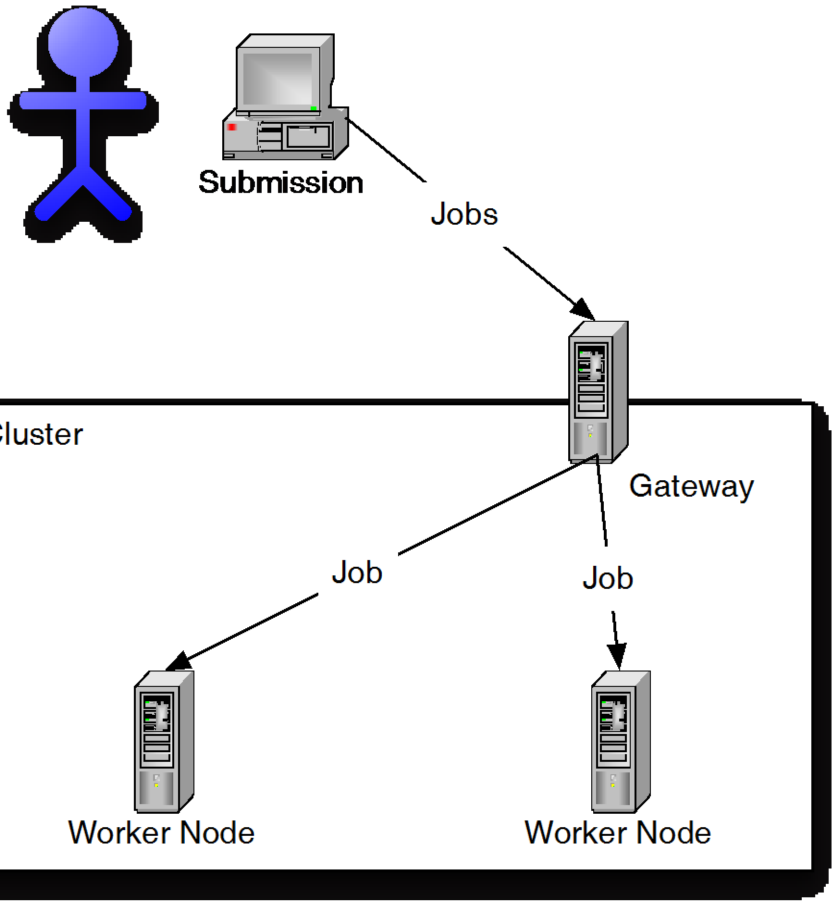
13,575,000	Jobs
58,625,000	CPU Hours
72,259,000	Transfers
35,724	TB Transferred

In the last Year

195,889,000	Jobs
719,558,000	CPU Hours
645,850,000	Transfers
322,462	TB Transferred

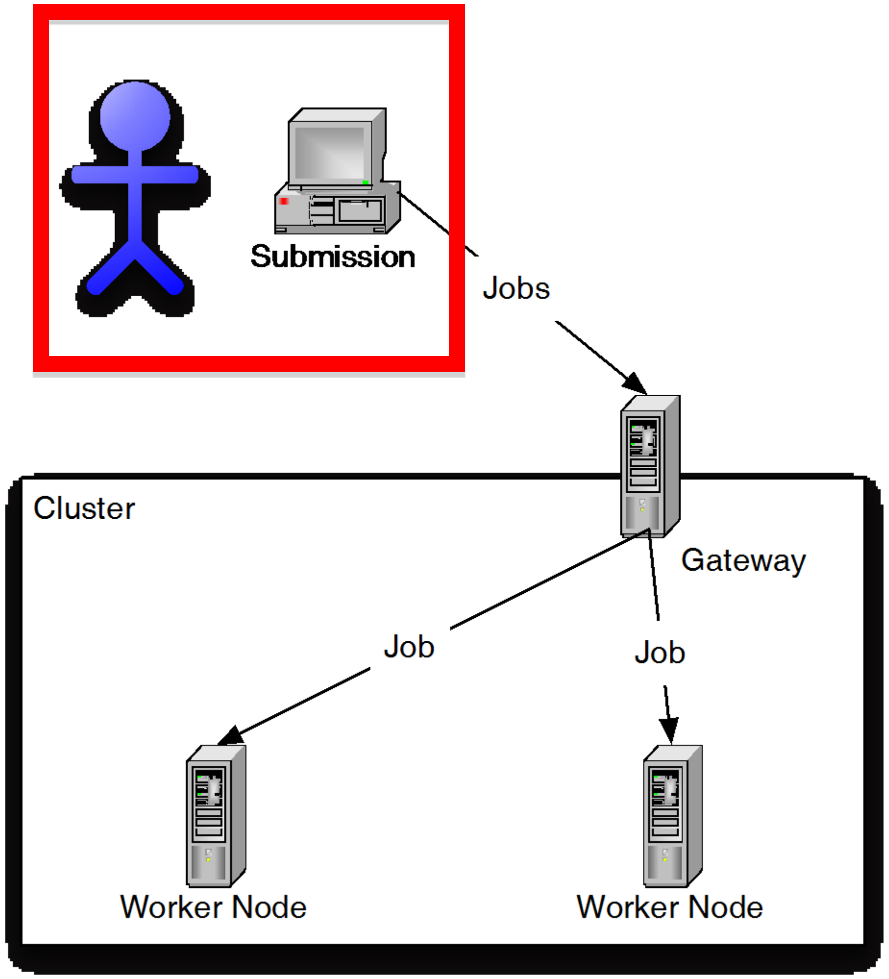
Typical Site in OSG

- User creates executable and submits jobs
- Job translated from local submission to generic submission language.
- Jobs translated from generic submission language to cluster specific.
- Jobs execute on the remote worker nodes



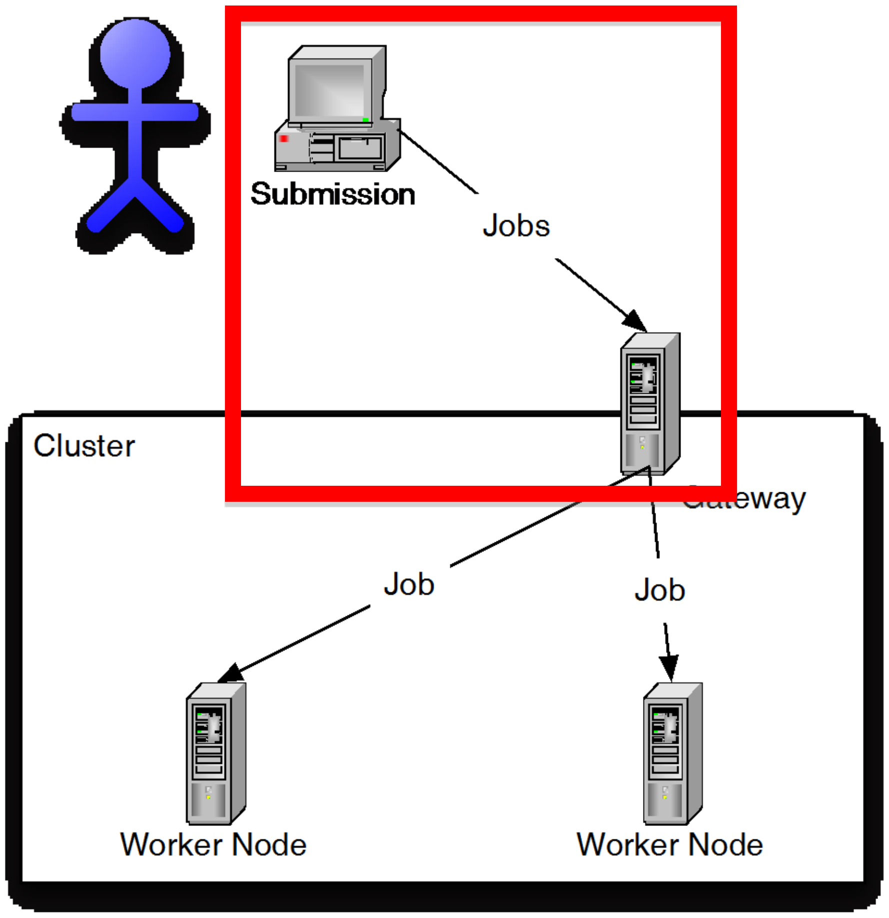
Typical Site in OSG

- User creates executable and submits jobs
- Job translated from local submission to generic submission language, sent to gatekeeper
- Jobs translated from generic submission language to cluster specific.
- Jobs execute on the remote worker nodes



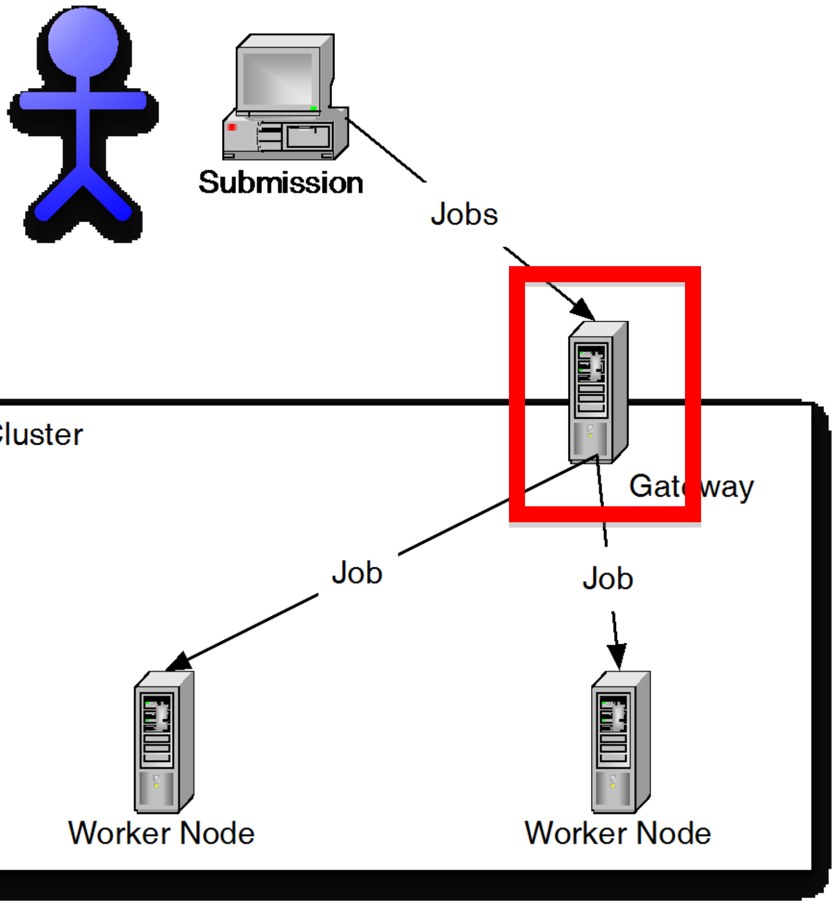
Typical Site in OSG

- User creates executable and submits jobs
- Job translated from local submission to generic submission language, sent to gatekeeper
- Jobs translated from generic submission language to cluster specific.
- Jobs execute on the remote worker nodes



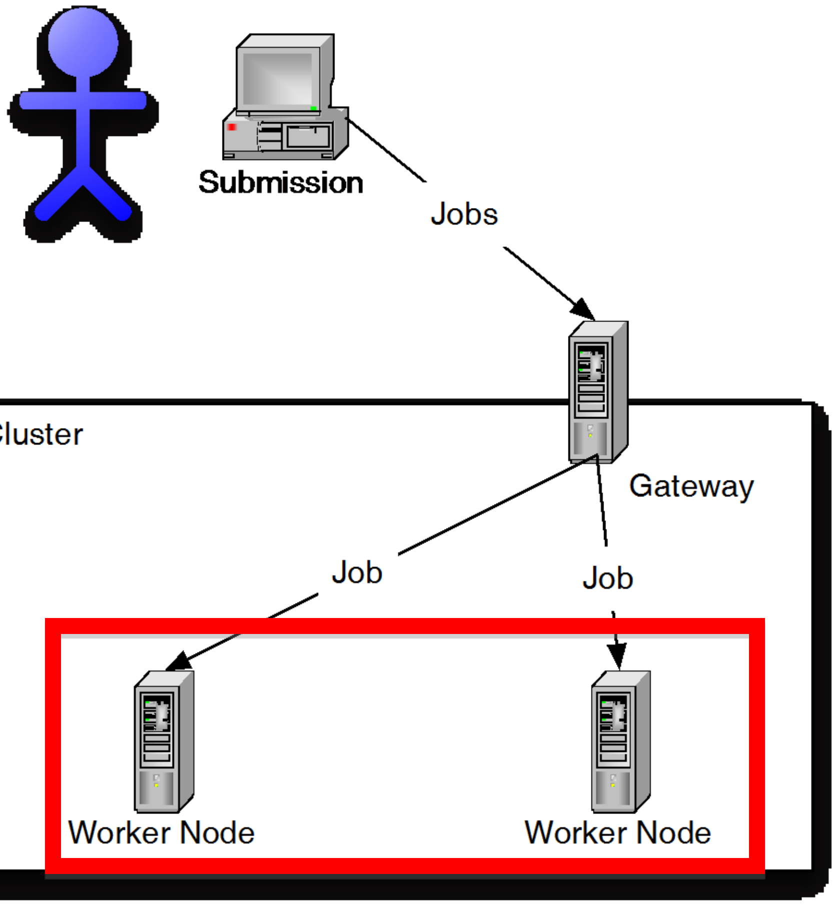
Typical Site in OSG

- User creates executable and submits jobs
- Job translated from local submission to generic submission language, sent to gatekeeper
- Jobs translated from generic submission language to cluster specific.
- Jobs execute on the remote worker nodes



Typical Site in OSG

- User creates executable and submits jobs
- Job translated from local submission to generic submission language, sent to gatekeeper
- Jobs translated from generic submission language to cluster specific.
- Jobs execute on the remote worker nodes



Typical Site Sizes

- Nebraska – 16k cores on OSG
- Purdue – 20k+ cores
- UCSD, MIT, Caltech – 3k cores
- Wisconsin – 10k cores

- We’re never going to get all of these cores (we don’t own any), but we could get some small fraction.

Outline

Motivation

- Create a Opportunistic Hadoop framework for researchers

- Researchers can run on the OSG for free

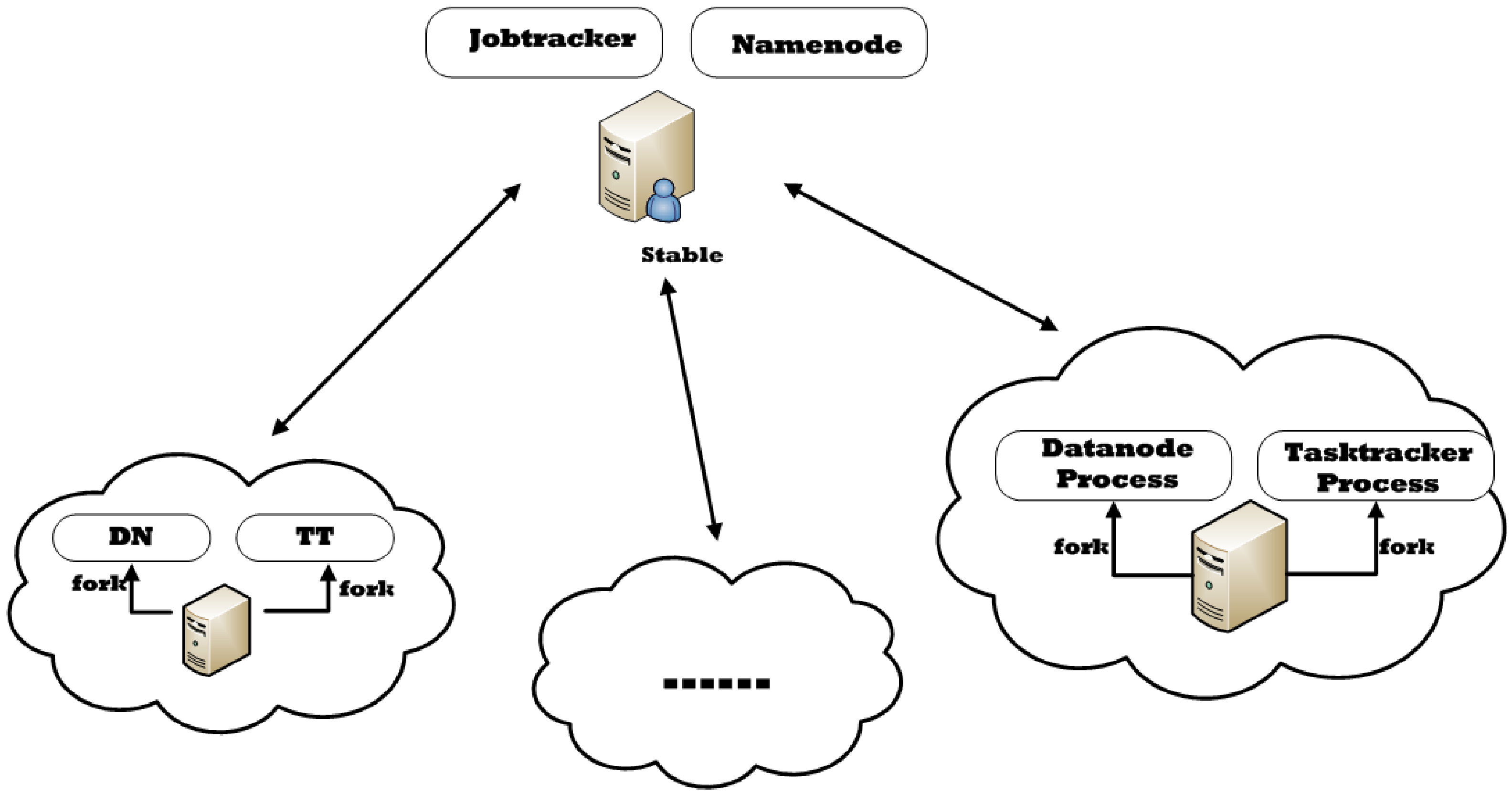
- Free Hadoop execution on the OSG

Motivation

- Major infrastructure already existing to run Scientific Computing
  - Grids (US: Open Science Grid, Europe: European Grid Initiative) can run high throughput, fault tolerant, data intensive computing.
  - Super Computers managed by XSEDE (formally Teragrid) offer petaflops of HPC to scientists.
  - These resources are free (XSEDE has allocations) for researchers.

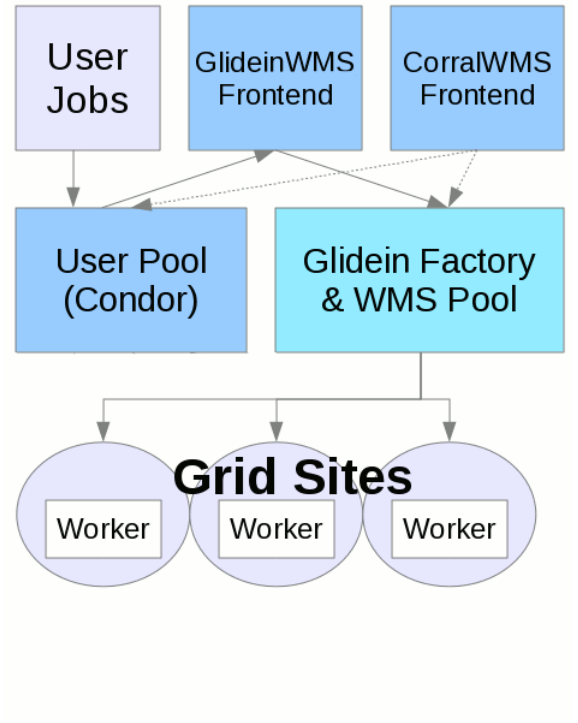
Architecture

- Hadoop on the Grid



Sending Jobs to the OSG

- Use GlideinWMS (tinyurl.com/glideinwms) for grid submission



Job Description

- Each Hadoop worker node is a packaged into a grid job.

- The grid jobs describe the requirements of the hadoop worker node
  - Public IP address (for internode communication)
  - Disk space for data node

Sending Jobs to the OSG

1. Initialize the OSG operating environment
2. Download the Hadoop worker node executables
3. Extract the worker node executables and set late binding configurations
4. Start the Hadoop daemons
5. When the daemons shut down, clean up the working directory.

Sending Hadoop Jobs to OSG

- Jobs are sent with small wrapper to download Hadoop executable.
- Hadoop executable (75MB) is cached at each site in their HTTP caches
- Wrapper modifies the Hadoop configuration and starts Hadoop on the remote worker node.

Hadoop on the Grid

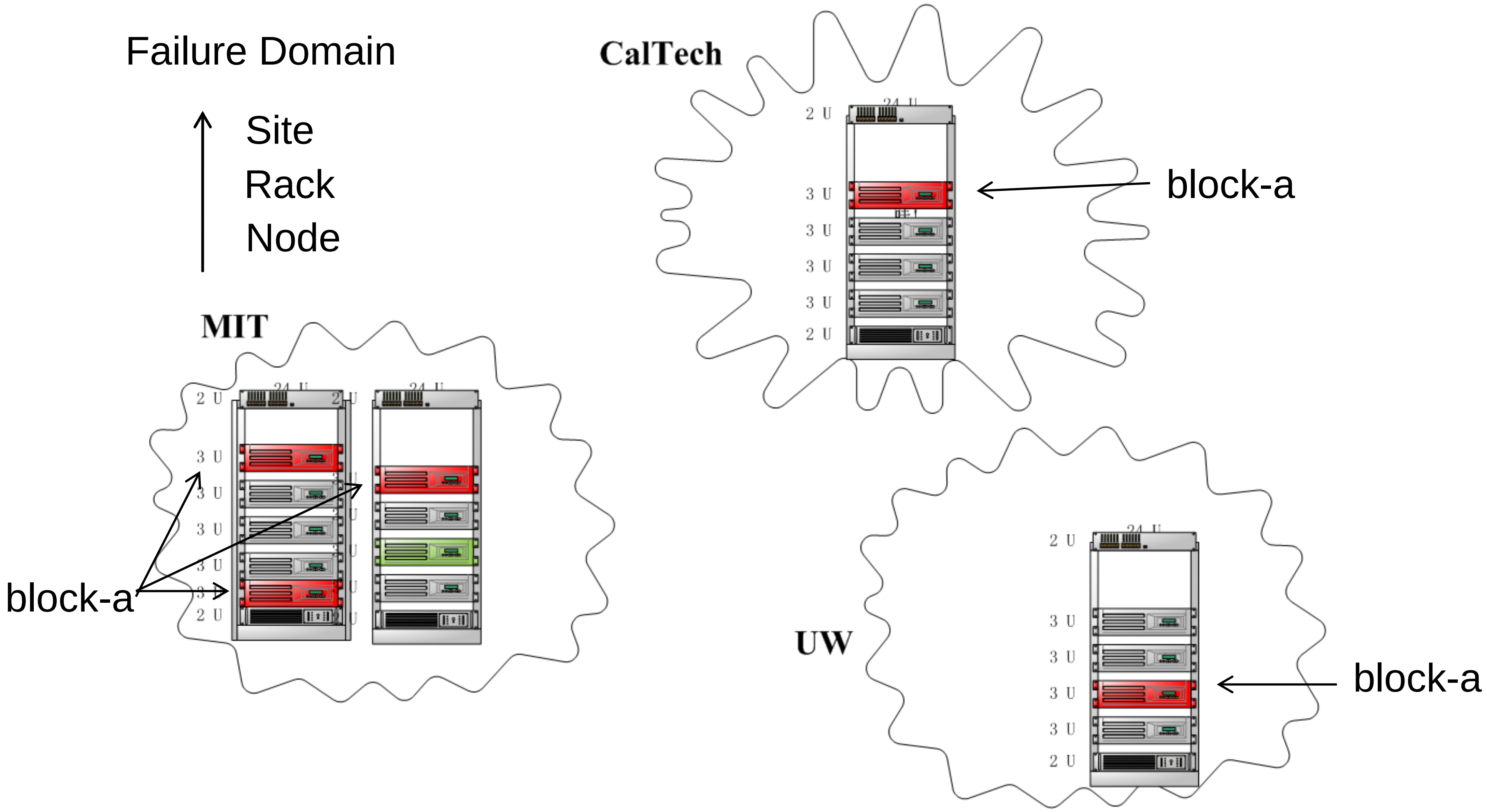
- The grid job will start the Hadoop datanode and tasktracker on the worker node.
- Since we are running opportunistically, we can be preempted at any time
- We must increase the robustness of Hadoop to survive site preemption and failures

Rack Awareness

- Rack awareness affects both job placement and data placement.
  - Tasks are targeted to racks that have the data
  - Data is copied to a node on the same rack, and at least one other rack for fault tolerance

- We want to use Rack awareness to shield us from entire cluster failures

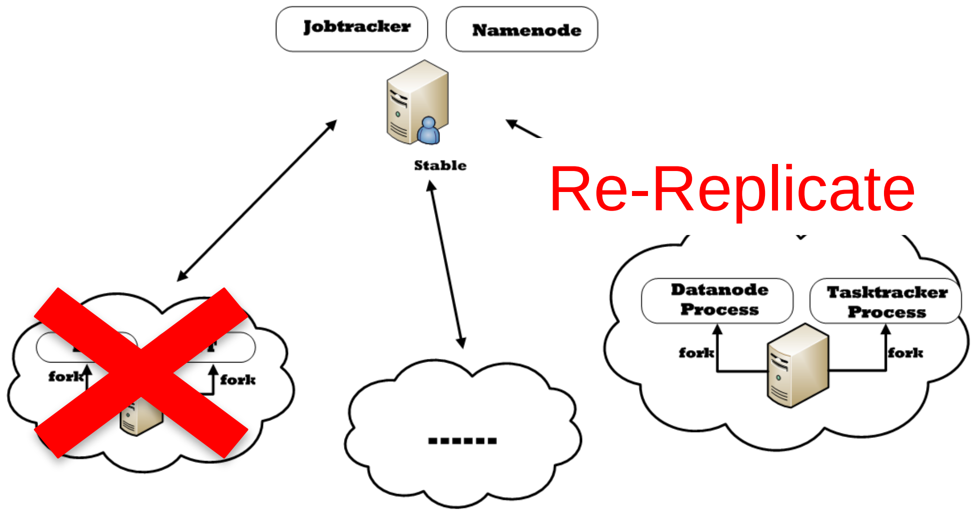
Data availability & Fault-tolerance



Rack Awareness

- In our definition, we define a rack as a cluster

- Created a script that given an input, can discern what cluster the IP belongs to



Rack Awareness

- Rack awareness also improves our task placement
- In default Hadoop, it will place tasks on the same rack as the input data.
- With our extension from a rack as a site, data locality will consider site-local tasks

Outline

- Introduction
- Motivation
- HOG Architecture
- Experimental evaluation
- Conclusions
- Future work
- Questions

Experiment Setup

- Defined a common workload to compare a dedicated Hadoop cluster with HOG.
- Varied size of HOG, recording completion times for the workload.
- Compare response time between the cluster and HOG

Experiment Workload

- We want to compare performance between a dedicated Hadoop cluster and HOG
- We used the workload outlined by Facebook

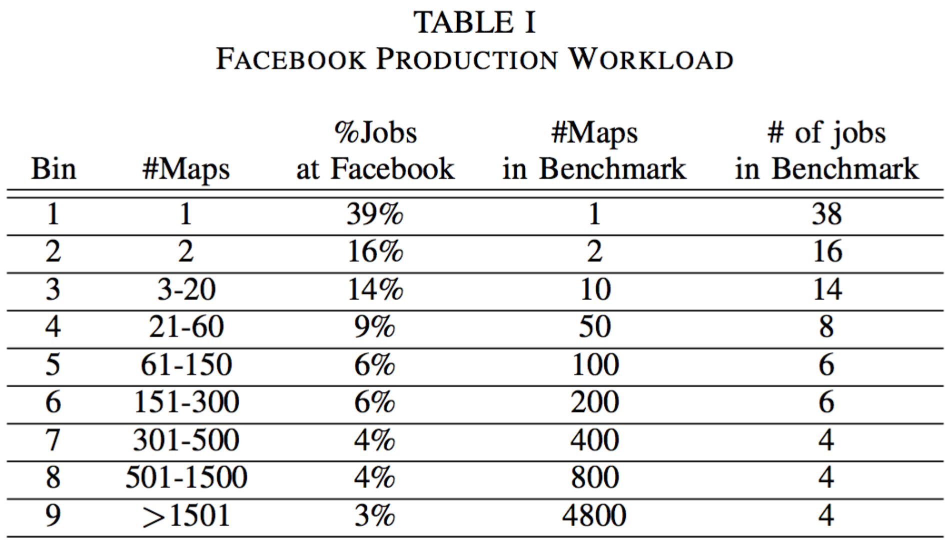


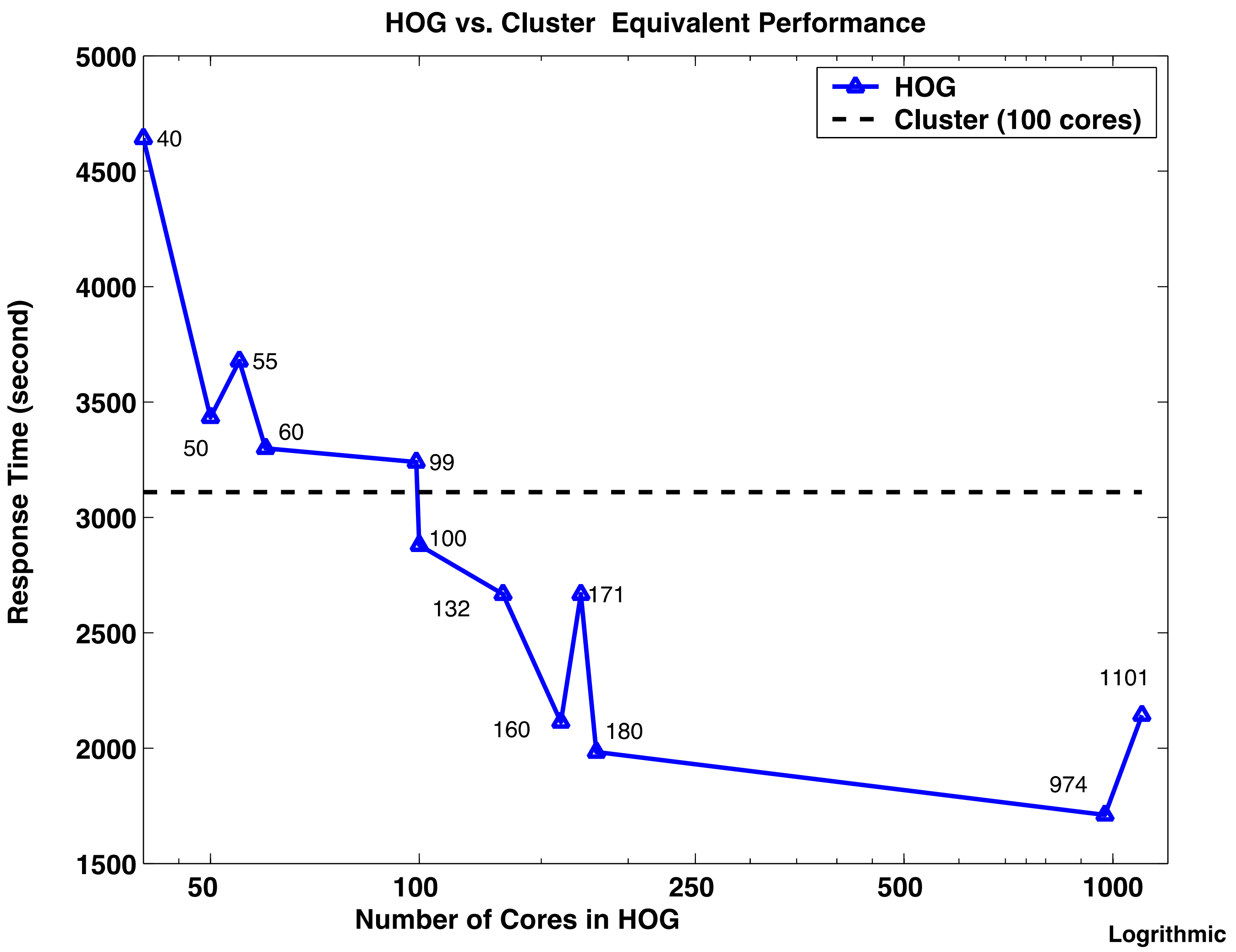
TABLE I
FACEBOOK PRODUCTION WORKLOAD

Bin	#Maps	%Jobs at Facebook	#Maps in Benchmark	# of jobs in Benchmark
1	1	39%	1	38
2	2	16%	2	16
3	3-20	14%	10	14
4	21-60	9%	50	8
5	61-150	6%	100	6
6	151-300	6%	200	6
7	301-500	4%	400	4
8	501-1500	4%	800	4
9	>1501	3%	4800	4

Experiment Setup

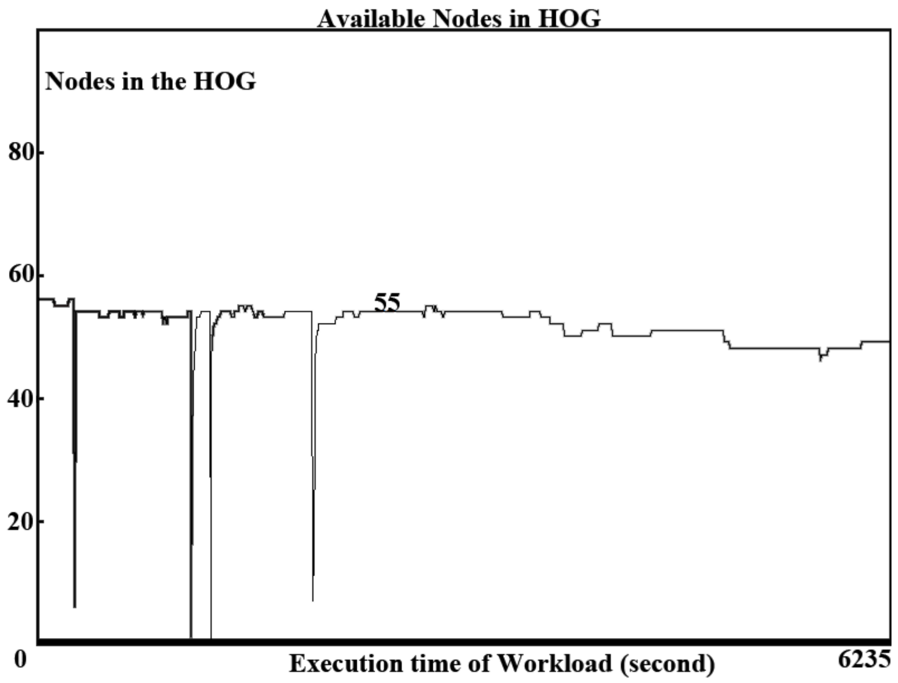
- Used a dedicated in-house cluster of 30 nodes, 100 cores.
- Submitted varying number of HOG nodes to the OSG to compare performance at different scales.
- Ran at each level 3 times, took average response time

Experimental Results - Scale



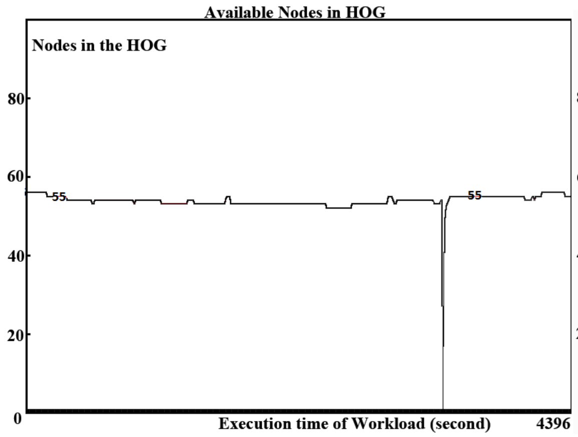
Experimental Results - Instability

- We had preemptions during our executions
- HOG system attempts to replace lost nodes

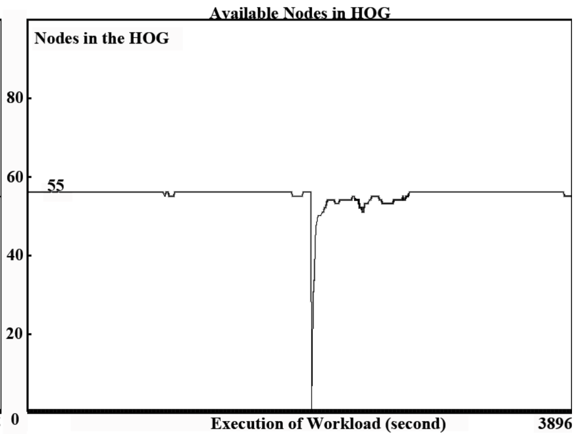


Experimental Results - Instability

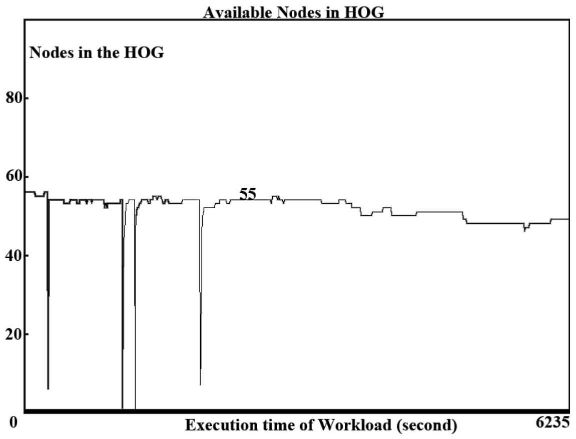
- Due to preemptions and failures, every run look very different



Moderately Stable



Stable



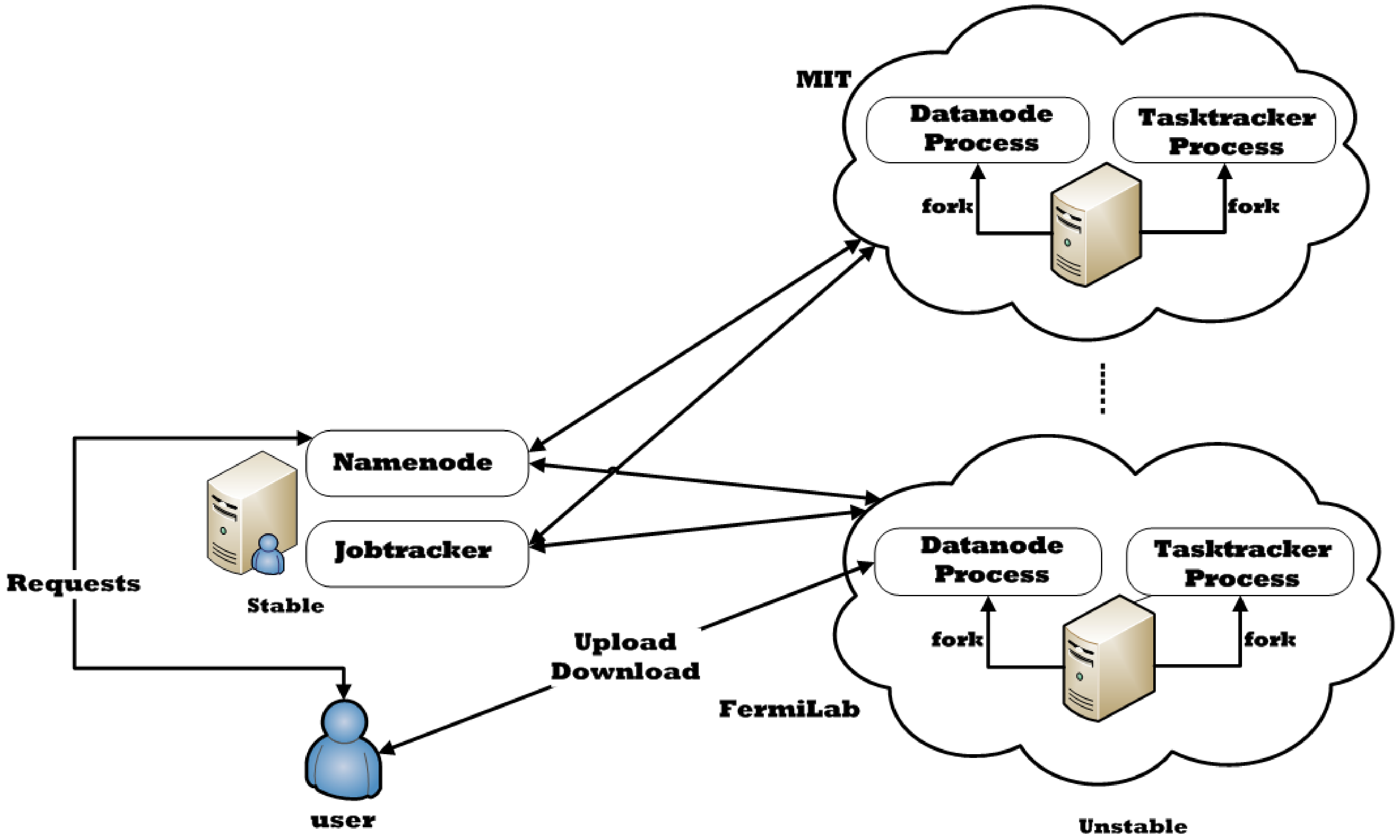
Unstable

Problems / Solutions

- During our evaluation found failure cases that Hadoop did not properly handle

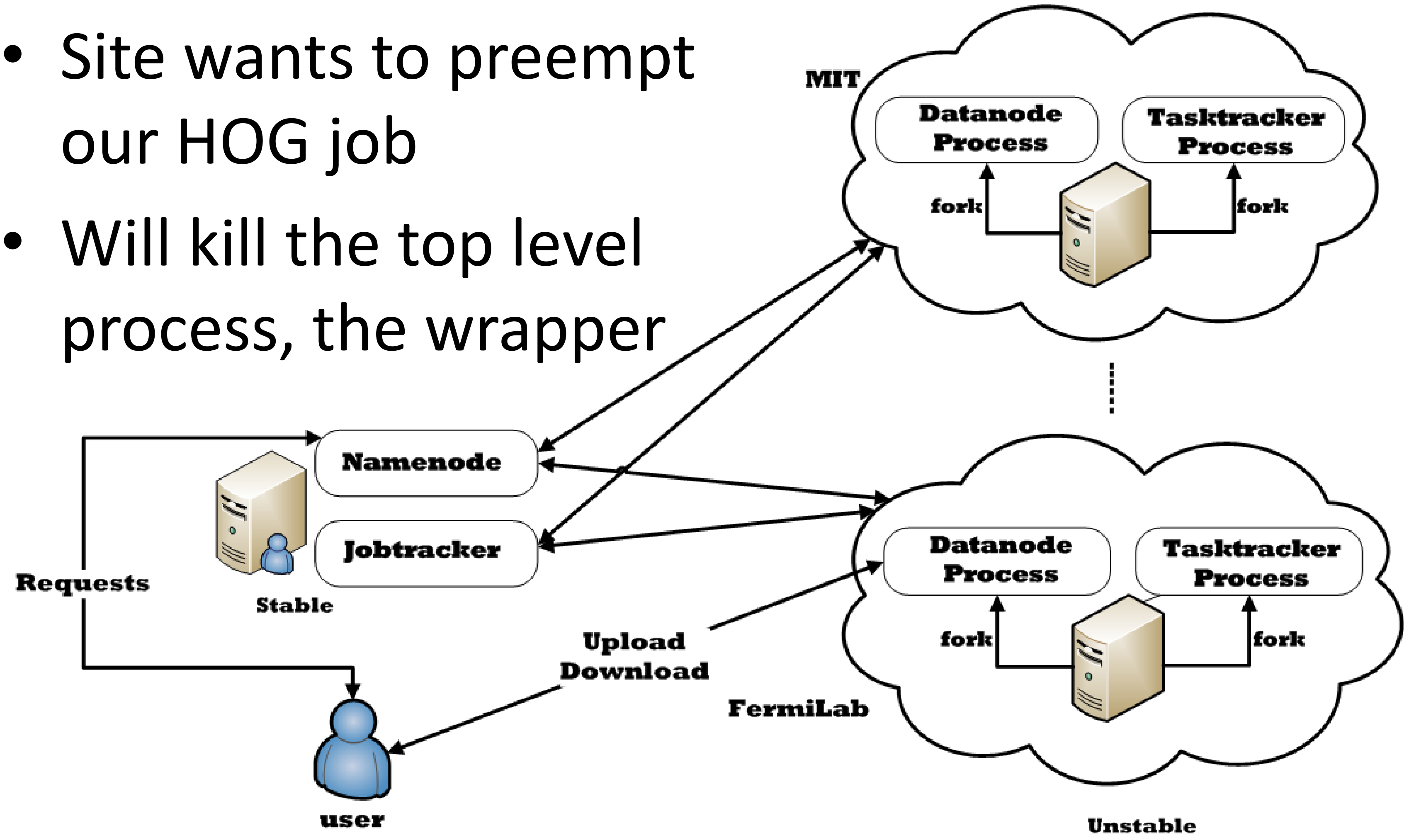
- Abandoned (zombie) datanodes
  - Datanode processes that stick around after the site has preempted the job
  - Can cause failures in user jobs due to missing input data on the datanodes

Zombie Datanodes



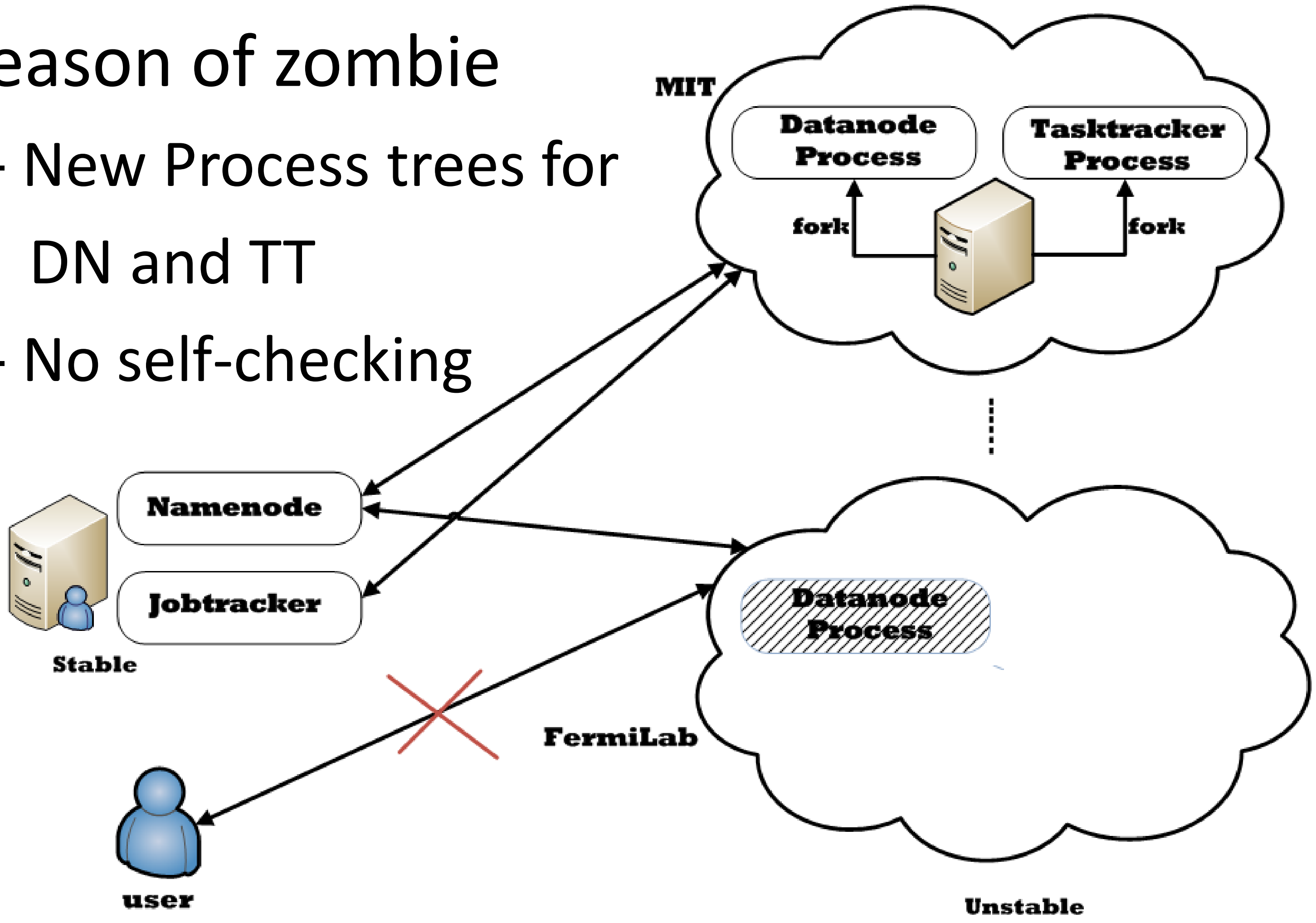
Zombie Datanodes

- Site wants to preempt our HOG job
- Will kill the top level process, the wrapper



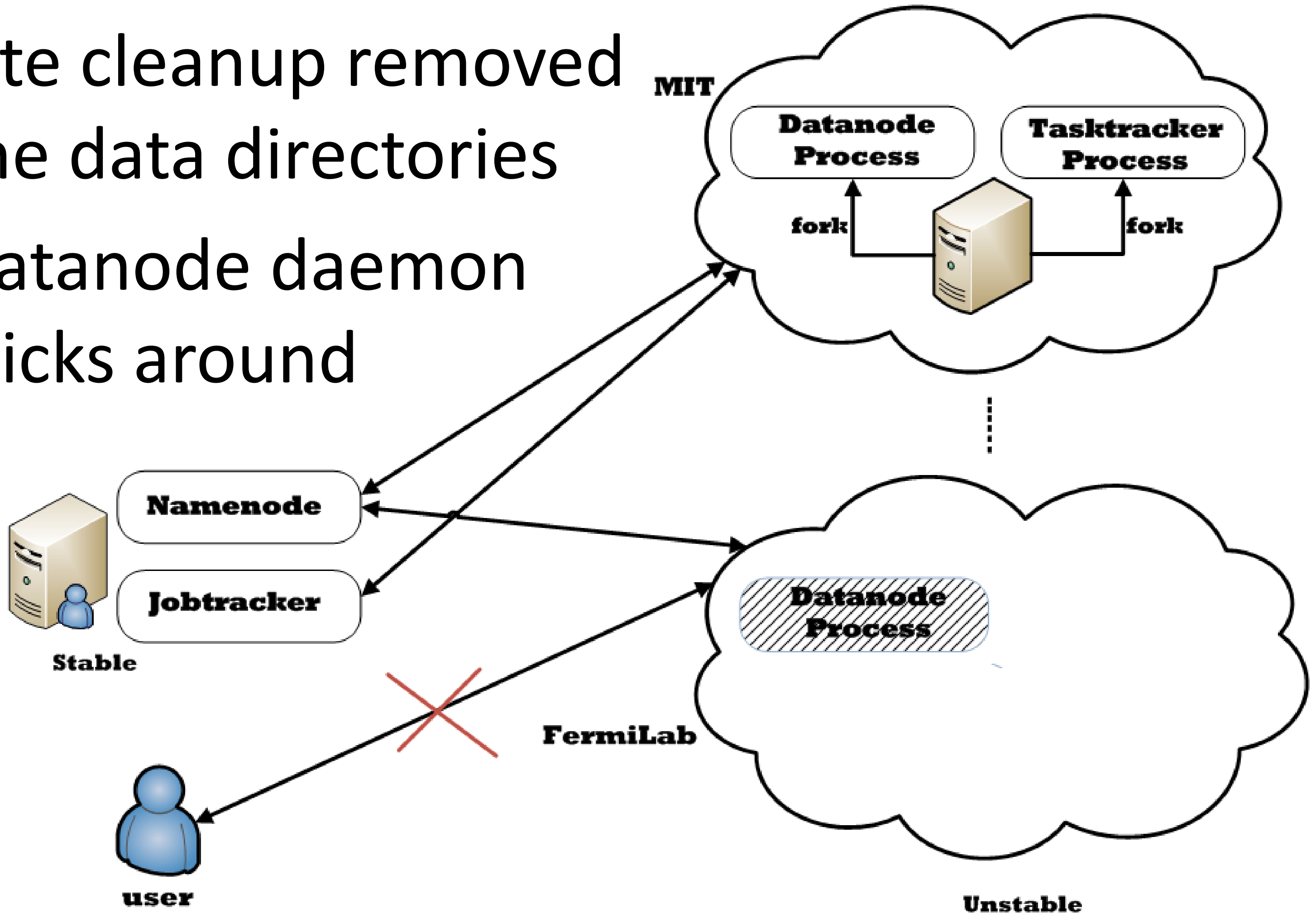
Zombie Datanodes

- Reason of zombie
  - New Process trees for DN and TT
  - No self-checking



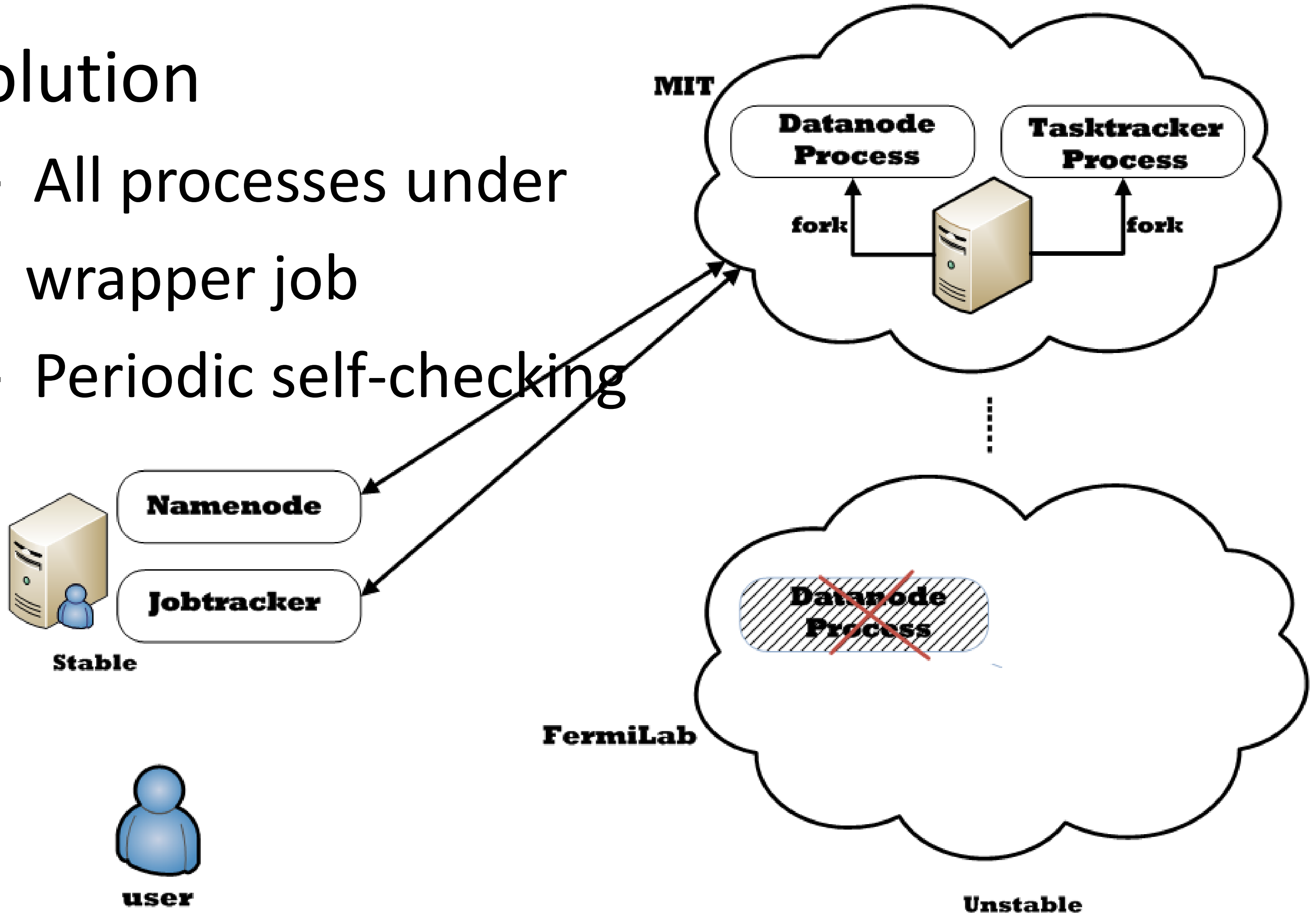
Zombie Datanodes

- Site cleanup removed the data directories
- Datanode daemon sticks around



Zombie Datanodes

- Solution
  - All processes under wrapper job
  - Periodic self-checking



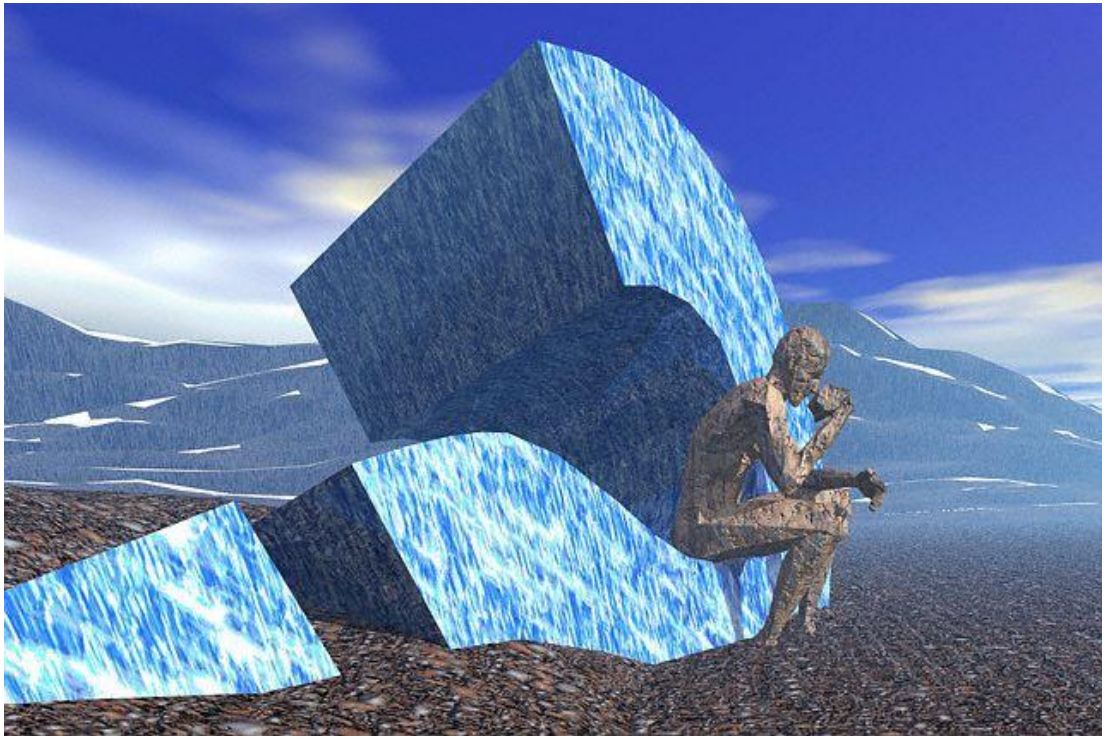
Conclusions

- Created and evaluated a Hadoop framework on the OSG
- Since the framework is run on the OSG, it is free to researchers.
- Site awareness added to increase reliability
- Showed equivalent performance compared with a dedicated Hadoop Cluster

Future work

- There is still much work to be done.
  - Multiple duplicated tasks execution
  - Data blocks replication and movement monitoring during HOG scaling
- Improved security
  - Since transfers are happening over the WAN, we need to authenticate data nodes with each other
  - Can use grid certificates to provide PKI security between nodes

Questions



Backup Slides

Sending Jobs to the OSG

- GlideinWMS Submission File

```
universe = vanilla
requirements =  GLIDEIN_ResourceName =?= "FNAL_FERMIGRID" ||
GLIDEIN_ResourceName =?= "USCMS-FNAL-WC1" || GLIDEIN_ResourceName =?=
"UCSDT2" ||  GLIDEIN_ResourceName =?= "AGLT2" || GLIDEIN_ResourceName
=?= "MIT_CMS"
executable = wrapper.sh
output = condor_out/out.$(CLUSTER).$(PROCESS)
error = condor_out/err.$(CLUSTER).$(PROCESS)
log = hadoop-grid.log
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
OnExitRemove = FALSE
PeriodicHold = false
x509userproxy = /tmp/x509up_u1384
queue 1000
```