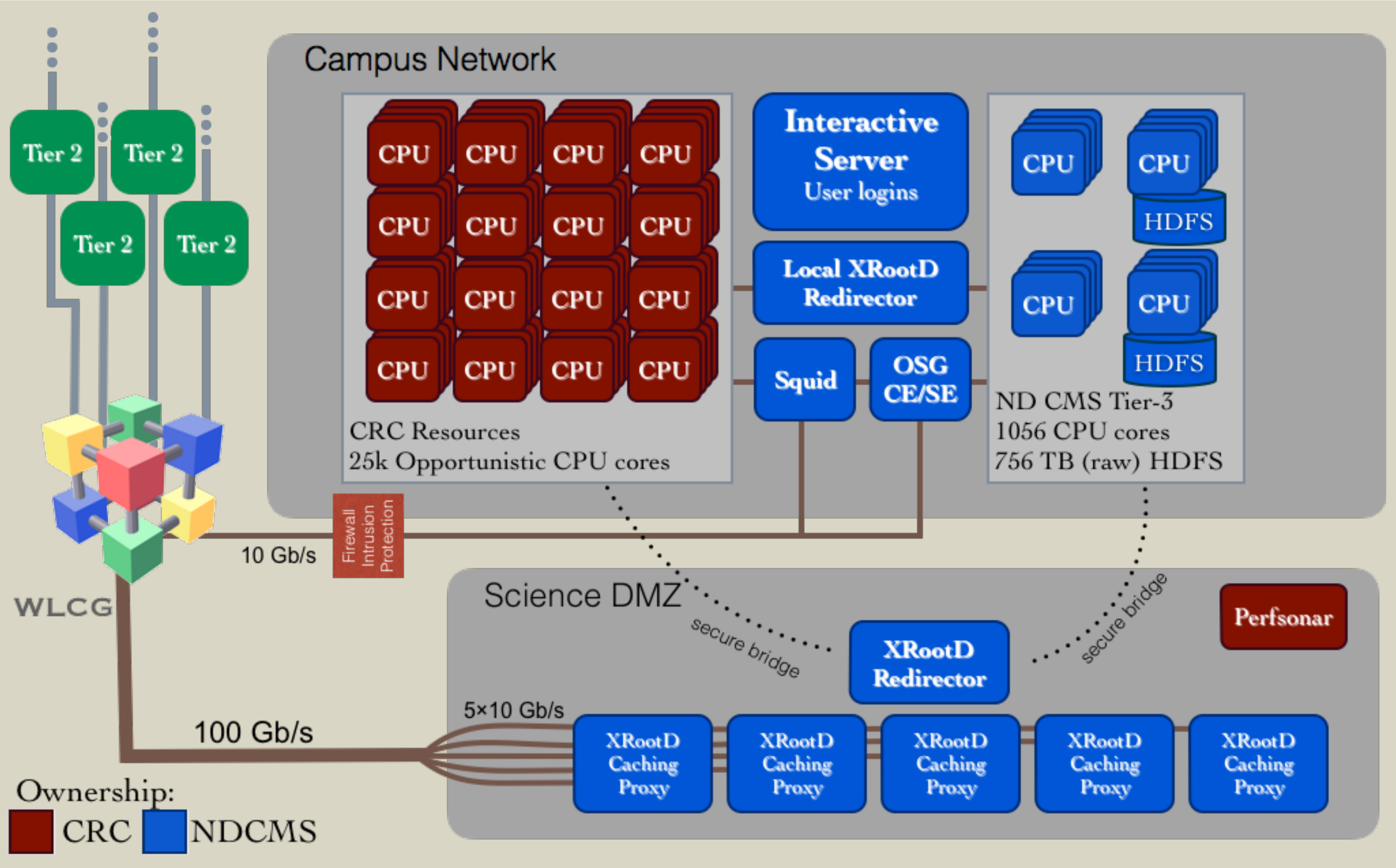
Scaling Up a CMS Tier-3 Site with Campus Resources and a 100 Gb/s Network Connection: What Could Go Wrong?

Matthias Wolf, Anna Woodard, Ben Tovar, Kenyi Hurtado Anampa, Paul Brenner, Kevin Lannon, Michael Hildreth, Douglas Thain

The Notre Dame CMS group operates a modest-sized Tier-3 site. Through the Center for Research Computing, Notre Dame researchers have opportunistic access to roughly 25k CPU cores and 100 Gb/s WAN. We undertook to use these resources for a wide range of CMS computing tasks from user analysis through large-scale Monte Carlo production. We will discuss the challenges inherent in effectively utilizing CRC resources for these tasks and the solutions deployed to overcome those challenges. We will also discuss current performance and future refinements as well as interactions with the broader CMS computing infrastructure.

Center for Research Computing (CRC)

The CRC serves Notre Dame's computational infrastructure needs. They administer over 25,000 compute cores on approximately 2,000. The resources comprise both shared resources (~1/3) and faculty-owned systems (~2/3). Two resource schedulers run simultaneously: UGE and HTCondor. Faculty owners have immediate access to their systems via UGE which pre-empts & evicts opportunistic jobs scheduled via HTCondor.



ND CMS Tier 3

Originally a self-contained cluster, ND's Tier-3 now serves as bridge into CRC Resources, running the necessary services (XRootD, Squid, OSG).

XRootD Proxy/Cache Configuration

CMS data files are globally distributed via XRootD. Though the campus worker nodes are not directly part of the 100 Gb/s network, they can access it via proxy servers. Caching of CMS data files on the local proxy servers allows faster access by worker nodes and reduces incoming network traffic.

100 Gbps CMS Science DMZ

ND has constructed a Science DMZ connected to the nationalI2 & ESnet networks at 100Gb/s.

Challenges

DMZ and Network

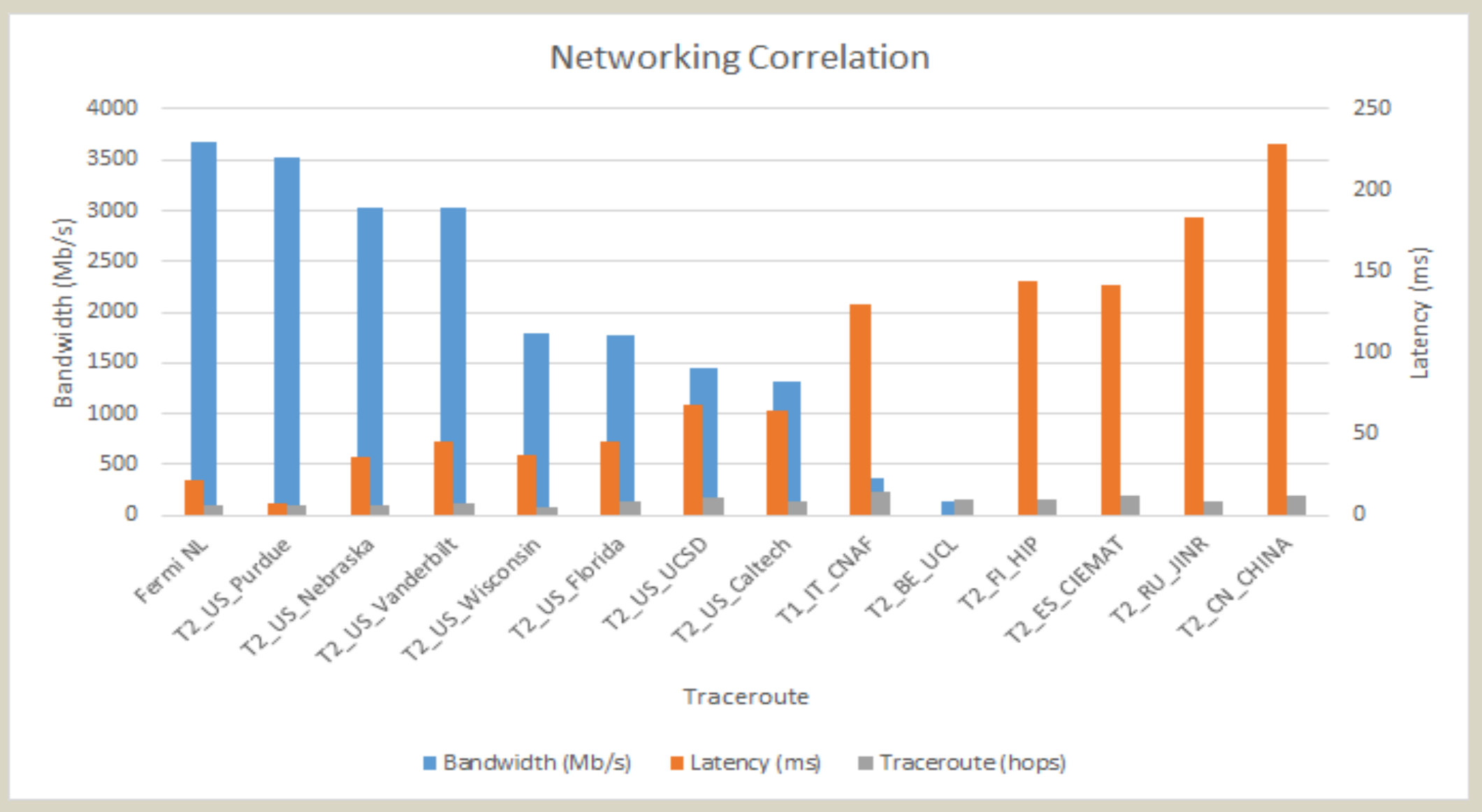
Workflow Management

100 Gb/s Network

Challenges:

- Tuning Single Stream (baseline)
- Tuning Aggregate Streams (realistic)
- Network Contention/Variance
- DMZ Complexity
- Routing for Security and Performance

We have begun testing network performance between ND and CMS T1 and T2s. A 10Gb tuned PerfSonar node runs automated tests to identify single data stream bottlenecks and baseline values. One particular challenge arises because the goal is to aggregate multiple 10 Gb/s connections to fill the 100 Gb/s network link with transfers from multiple sites. It can be hard to spot bottlenecks that are ≥ 10 Gb/s. Example: For one T2 site, found a 10 Gb/s bottleneck in what was supposed to be a fully 100 Gb/s route because one link carrying half the traffic was accidentally not upgraded from 10 Gb/s to 100 Gb/s.



Measurements using Perfsonar of Bandwidth, Latency, and # of hops. There is a loose correlation between bandwidth and latency, but there is enough variation from one site to the next to suggest that further debugging of network connections could be beneficial.

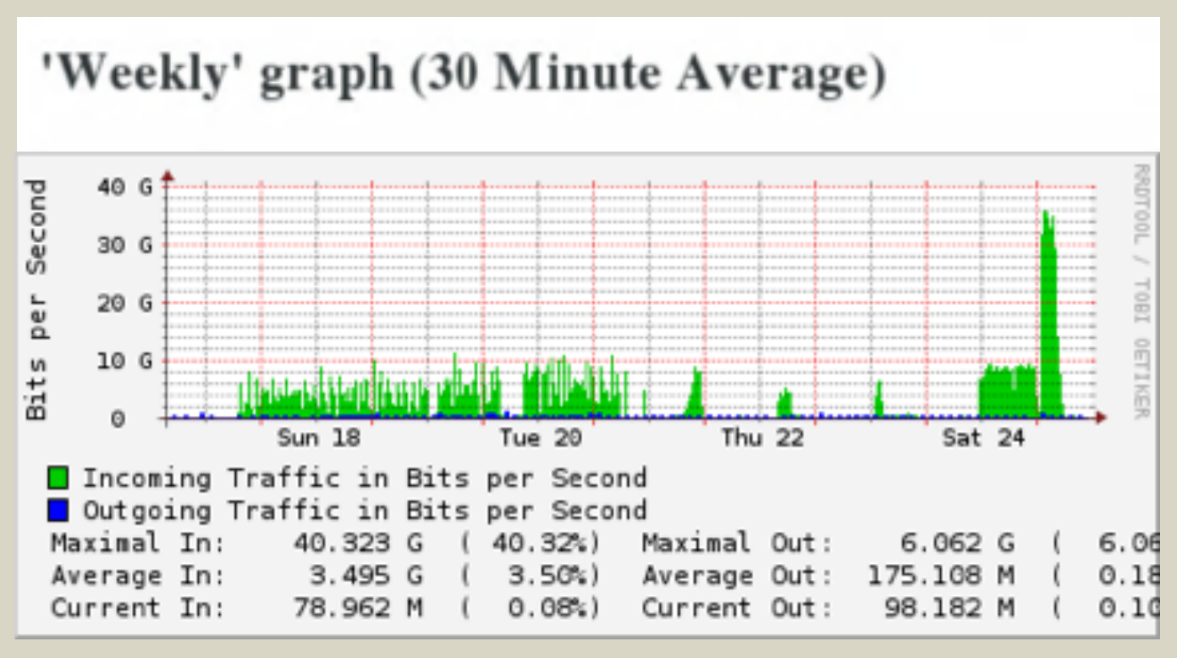
XRootD Proxy/Cache Servers

Challenges:

- A single cache proxy server is not enough!. The high CPU, memory load and high disk I/O this demands makes having multiple servers more desirable, so load balancing is important, a cluster management daemon is run to achieve this (this is part of the XRootD components).
- This is the first time a CMS Tier center uses XRootD caching proxy configuration at this scale. A collaboration with XRootD developers started months ago, substantially improving stability after 4 beta version releases (many thanks to Matevz Tadel (UCSD) and Andrew Hanushevsky (SLAC)!).

The servers talk to a single redirector through the Cluster Management System daemon (cmsd). We have started with 5 XRootD cache proxy servers with a 10 Gb/s duplex network link and 24 TB storage disk each.

The servers were tuned to deliver 6-7 Gb/s of data transfer for production workflows and we have achieved a sustained 30 minute average bandwidth over 35 Gbps so far. We aim to achieve 50 Gbps sustained in the coming year with further tuning and additional servers.

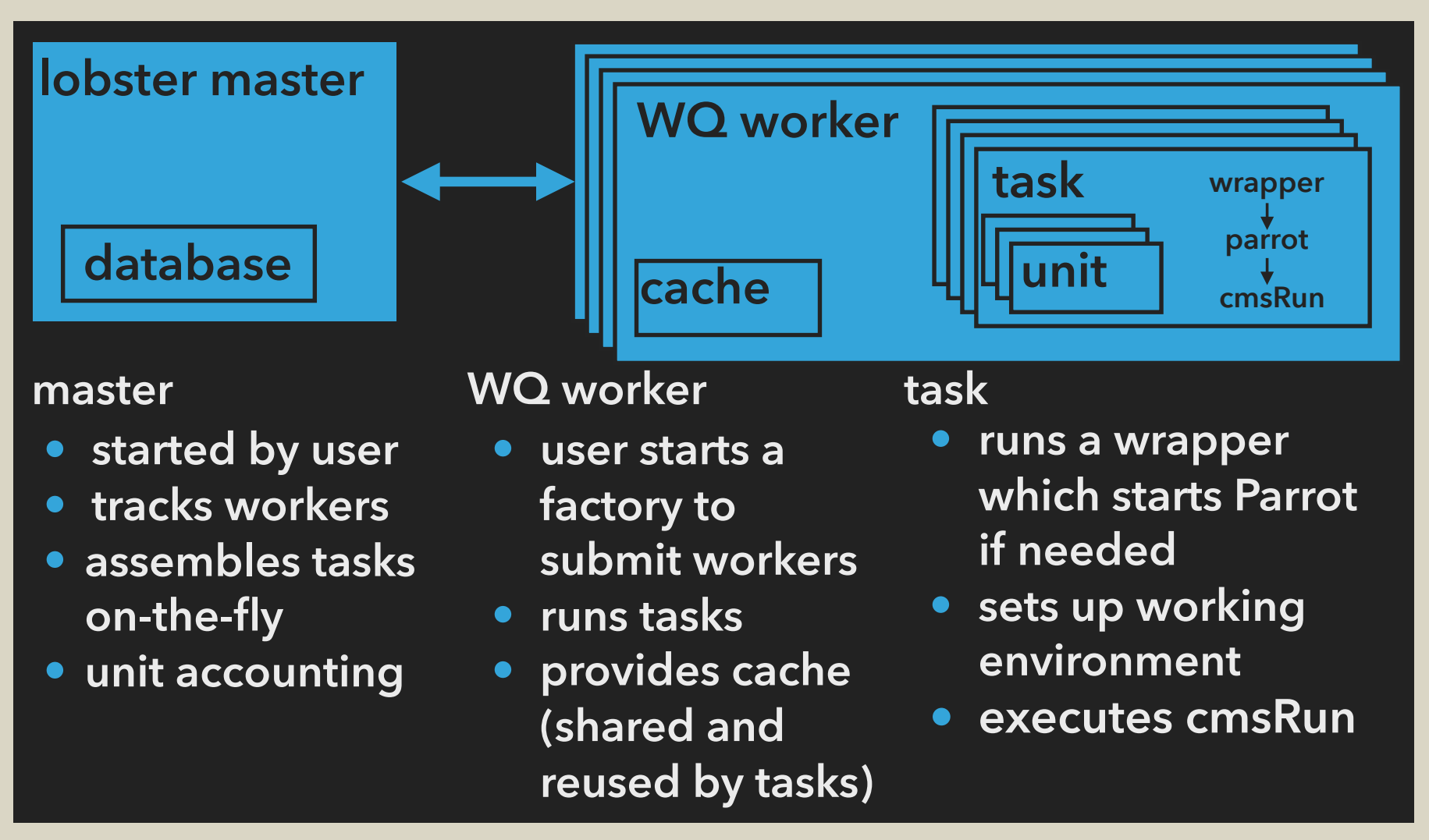


```
Maximal In:   40.323 G  ( 40.32%)  Maximal Out:    6.062 G  (  6.06
Average In:    3.495 G  (  3.50%)  Average Out:  175.108 M  (  0.18
Current In:   78.962 M  (  0.08%)  Current Out:   98.182 M  (  0.10
```

Average bandwidth traffic coming from the XRootD Servers

Lobster

Lobster is an opportunistic workflow manager, built on top of CCTools.



master
- started by user
- tracks workers
- assembles tasks on-the-fly
- unit accounting

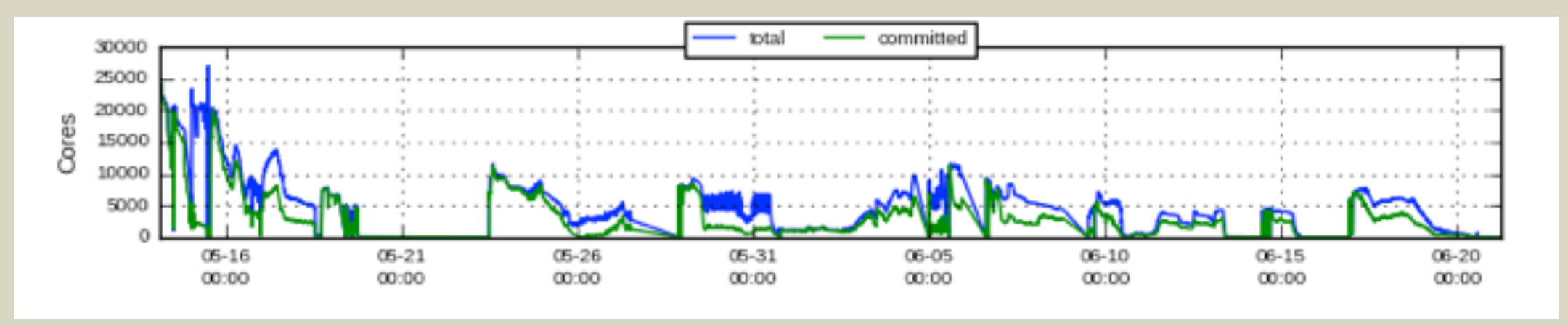
WQ worker
- user starts a factory to submit workers
- runs tasks
- provides cache (shared and reused by tasks)

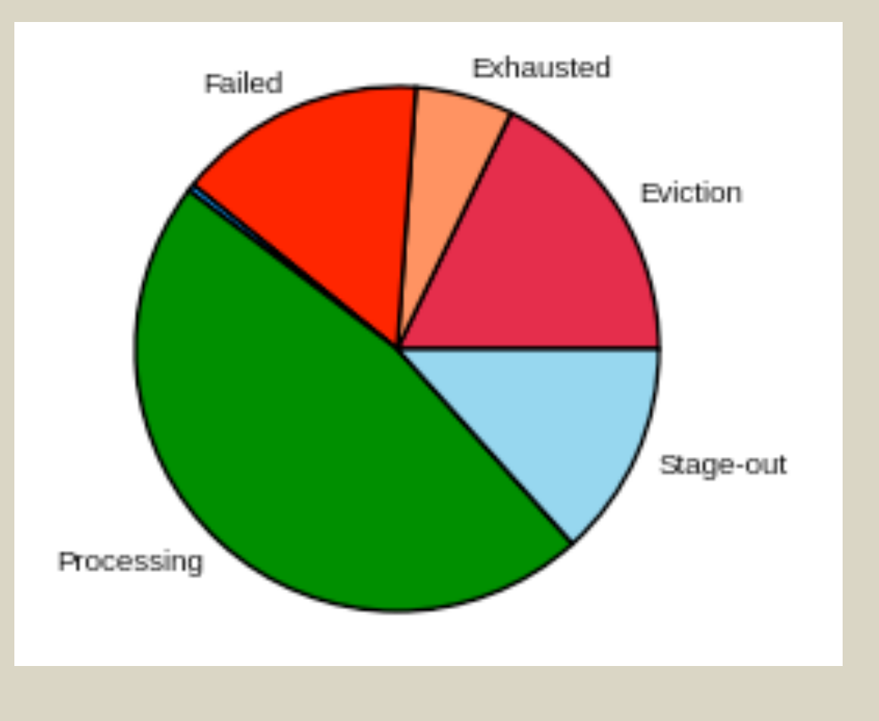
task
- runs a wrapper which starts Parrot if needed
- sets up working environment
- executes cmsRun

Learn more about Lobster!

Challenges and Solutions:

- Distribution of software to opportunistic nodes: Opportunistic resources accessed via CRC are not configured for CMS. CMS software environment is delivered over network via CVMFS + Parrot. Workers maintain cache of software that persists between tasks for lifetime of worker to minimize overhead in constructing CMS software environment.
- Can be preempted from CRC resources abruptly: When owners of CRC system request resources, opportunistic user is preempted with no warning. Lobster gives the ability to run short tasks to minimize preemption losses. Task output merged in separate step to produce conveniently sized output files.
- Resource utilization: Users often have only vague idea of resource needs of tasks from local testing. Lobster is built on CCTools components (Resource Monitor and Work Queue) that measures resource usage on running tasks and adjusts task resource needs accordingly. Resource Monitor also prevents tasks from crashing opportunistic resources by exceeding requested resource allocation. Evolving task resource needs are communicated back to user via monitoring plots, allowing user to tune worker resource allocations to best match task needs.
- Resiliency in the face of transient failures: The dynamic nature of opportunistic resources makes transient problems likely, resulting in potentially thousands of failed tasks. Lobster has built in redundancy for failure prone aspects of tasks: cascading fallback of input and output transfer mechanisms, automatic task retrying, and blacklisting of problematic worker nodes according to a user-configurable exit code list. Lobster tracks task failures and successes and handles all retries without user intervention, which is essential for running at scale.
- Scaling of Lobster master: As resource utilization scales to ~25k cores, it becomes challenging for the Lobster master to keep up with returning tasks and starting new tasks. We are currently exploring techniques to alleviate the load on the master, including use of WorkQueue foremen, delegating more accounting activity to the individual tasks, and optimizing database and filesystem access.





(Above) Plot showing number of cores used by Lobster, peaking around 25k cores. (Right) Accounting for time spent on processing versus time lost to job failures, jobs being killed for exhausting their resources, jobs being evicted, or delays in staging out that arising from the master becoming overloaded.

We are grateful to many people who assisted in constructing and troubleshooting this complex system; in particular, Jakob Blomer, Dan Bradley, and the CVMFS team; Dave Dykstra, Barry Blumenfeld and the Frontier team; Matevz Tadel, Andrew Hanushevsky and the XRootD team; Andrew Melo, Alan Tackett, Matt Heller and the Vanderbilt T2 Team; and Serguei Fedorov and the CRC staff.

This work was supported in part by National Science Foundation grant OCI-1148330 and a Department of Education GAANN fellowship.

UNIVERSITY OF NOTRE DAME