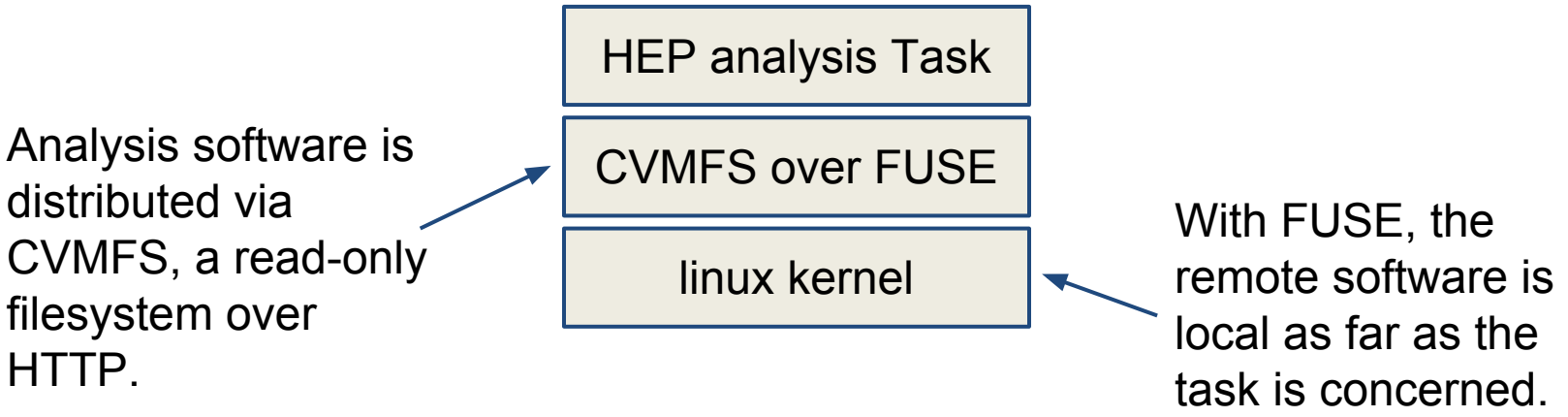
High-Energy Physics workloads on 10k non-dedicated opportunistic cores with Lobster

Anna Woodard, Matthias Wolf, et.al.
presented by Benjamin Tovar
Cooperative Computing Lab
http://ccl.cse.nd.edu

UNIVERSITY OF NOTRE DAME

Analyzing Data from the LHC

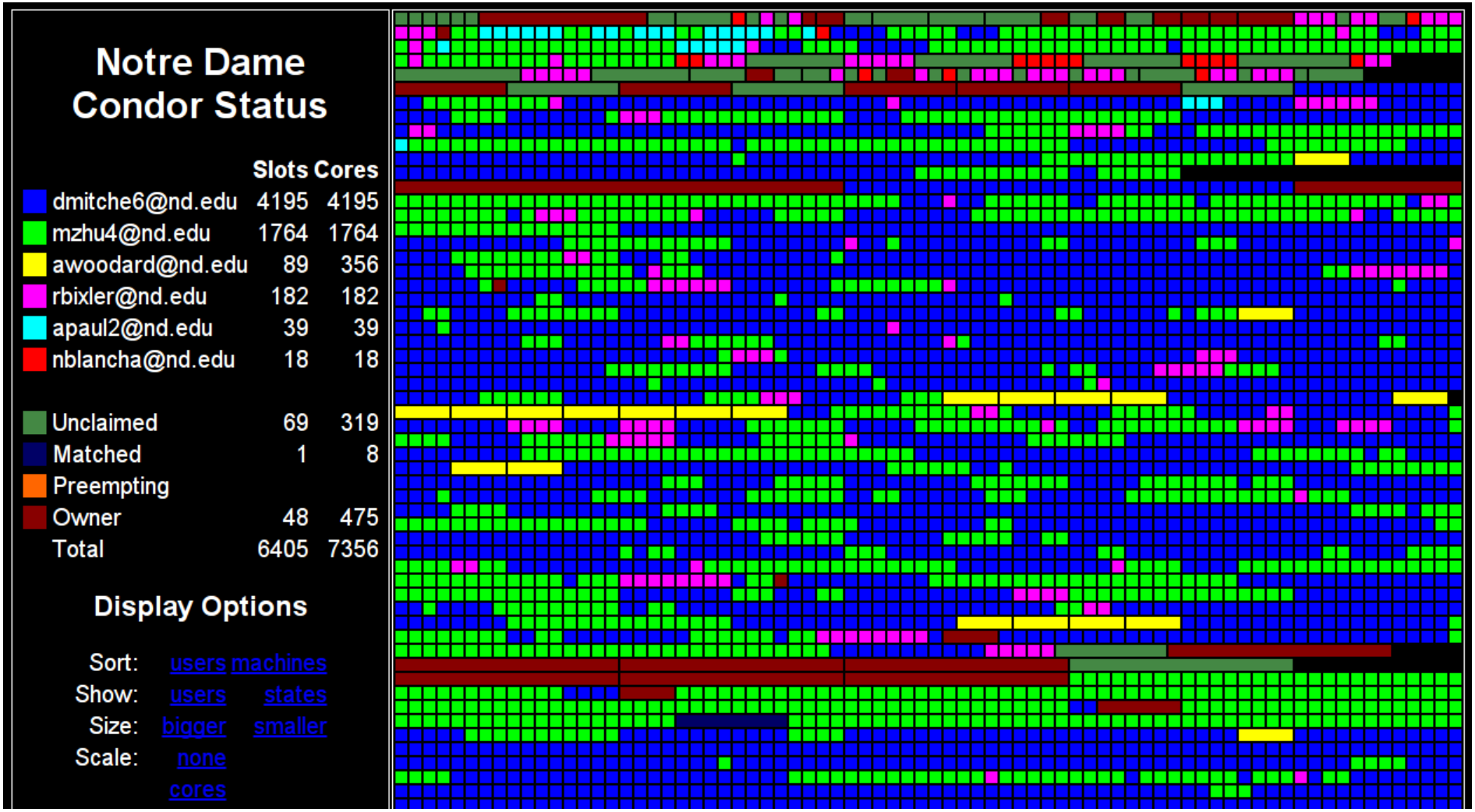
The High Energy Physics (HEP) community relies upon a global network of **dedicated** resources to analyze data from the Large Hadron Collider.



Notre Dame's happy opportunistic situation

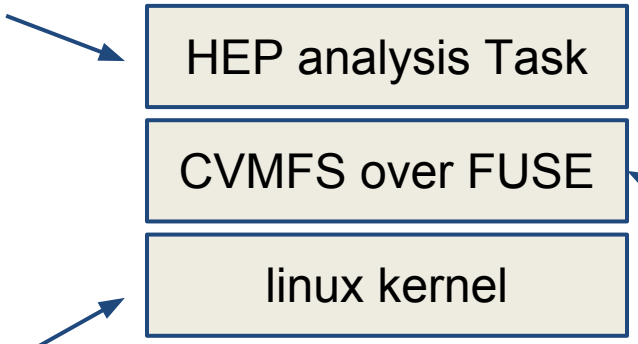
- ~21k cores at Notre Dame's Center for Research Computing (CRC)
- They belong to different individual PIs, but they are available through **condor** when not used by their owners.

condor.cse.nd.edu



from dedicated to opportunistic

Tasks running on opportunistic resources should be prepared for constant eviction.



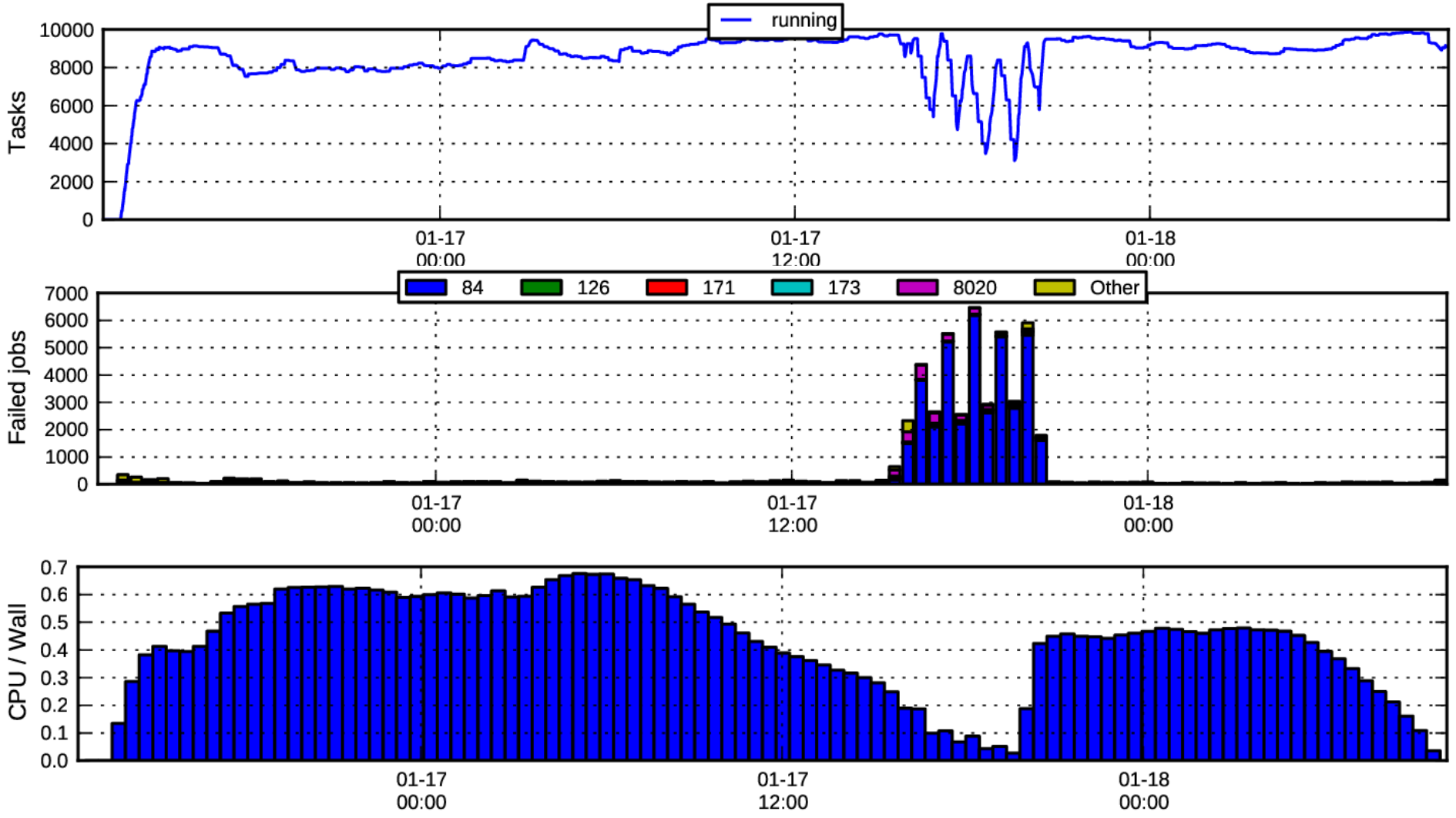
A FUSE module requires installation by the system administrator.

The biggest hurdle when going opportunistic.

For efficiency with several tasks running on the same node, CVMFS caches files.

If we are not careful, our bandwidth and disk space are no more...

preview of the results



ND CMS + CCTools + libCVMFS + CRC = Lobster

Anna Woodard
Matthias Wolf
Charles Mueller
Nil Valls
Kevin Lannon
Michael Hildreth

Ben Tovar
Patrick Donnelly
Peter Ivie
Douglas Thain

Jakob Blomer
Dan Bradley
Rene Meusel

Paul Brenner
Serguei Fedorov

Lobster is a system for deploying data intensive high-throughput application on non-dedicated resources

from dedicated to opportunistic

1. How a task may access CVMFS resources?
2. How can several tasks efficiently access the same data on a node?
3. How tasks can be sent to a computational node and managed?
4. How should tasks be decomposed to efficiently deal with eviction?
5. How the results of several tasks should be synthesized?

CCTools Philosophy

- Harness all the resources that are available: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware…
- **No special privileges required.**

A Quick Tour of the CCTools

- Open source, GNU General Public License.
- Compiles in 1-2 minutes, installs in $HOME.
- Runs on Linux, Solaris, MacOS, Cygwin, FreeBSD, …
- Interoperates with many distributed computing systems.
  - Condor, SGE, Torque, Globus, iRODS, Hadoop…
- Components:
  - **Work Queue** A lightweight distributed execution system.
  - **Parrot** A personal user-level virtual file system.
  - Makeflow A portable workflow manager.
  - Chirp A user-level distributed filesystem.

from dedicated to opportunistic

1. **How a task may access CVMFS resources?**
2. How can several tasks efficiently access the same data on a node?
3. How tasks can be sent to a computational node and managed?
4. How should tasks be decomposed to efficiently deal with eviction?
5. How the results of several tasks should be synthesized?

How a task may access CERN resources?

We use **parrot**

Parrot intercepts system calls and transforms them according to the requested service:

% **parrot_run** vi /anonftp/ftp.gnu.org/pub/README
% **parrot_run** ls **/cvmfs/**cms.cern.ch

parrot's dream use

parrot_run

a whole, unmodified workflow

parrot's practical use

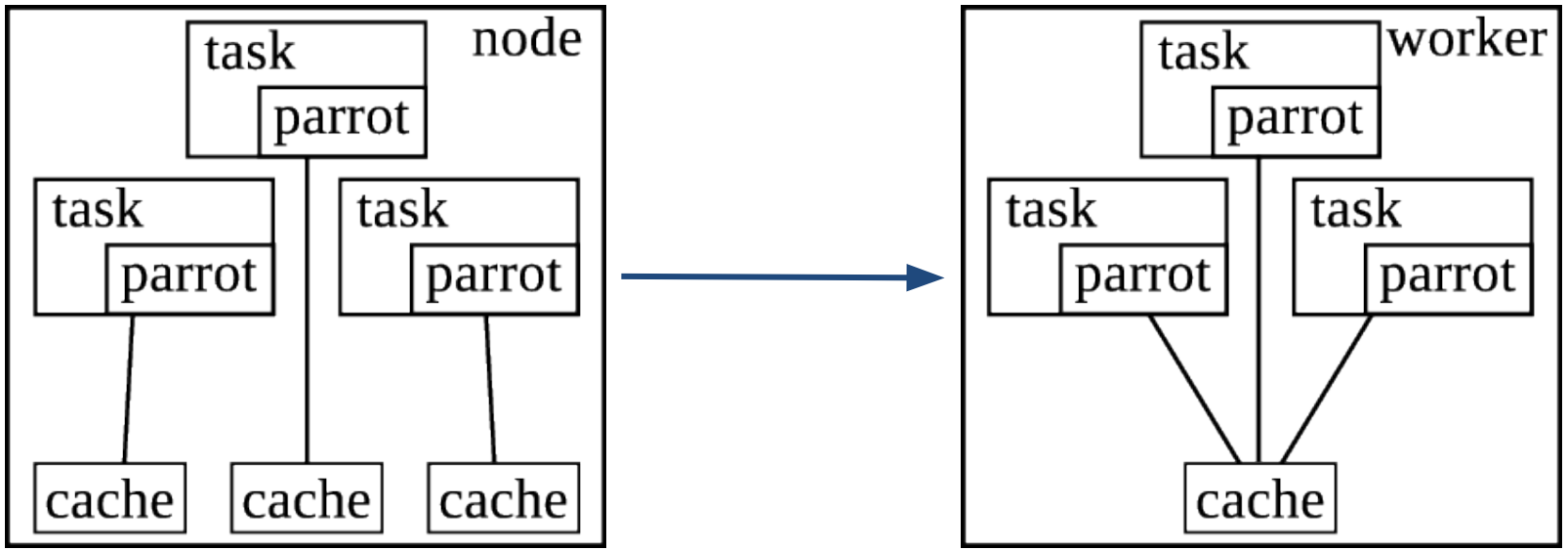
parrot has to mimic the kernel and de facto behaviour of glibc. It is a good way to discover the skeletons in the closet of the kernel, and thus it is better to restrict its use.

from dedicated to opportunistic

1. How a task may access CVMFS resources?
2. **How can several tasks efficiently access the same data on a node?**
3. How tasks can be sent to a computational node and managed?
4. How should tasks be decomposed to efficiently deal with eviction?
5. How the results of several tasks should be synthesized?

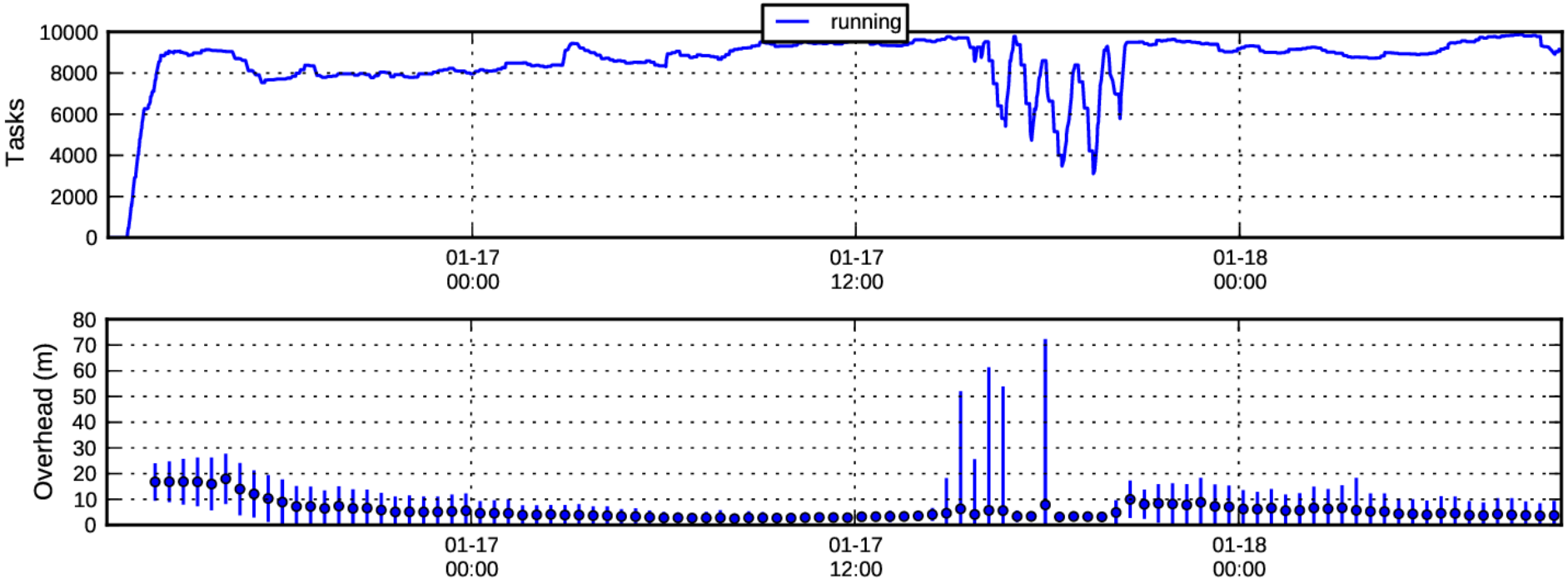
How can several tasks efficiently access the same data on a node?

We use **pilot jobs** (called **workers**) with condor, and libcvmfs' **alien cache** with parrot.



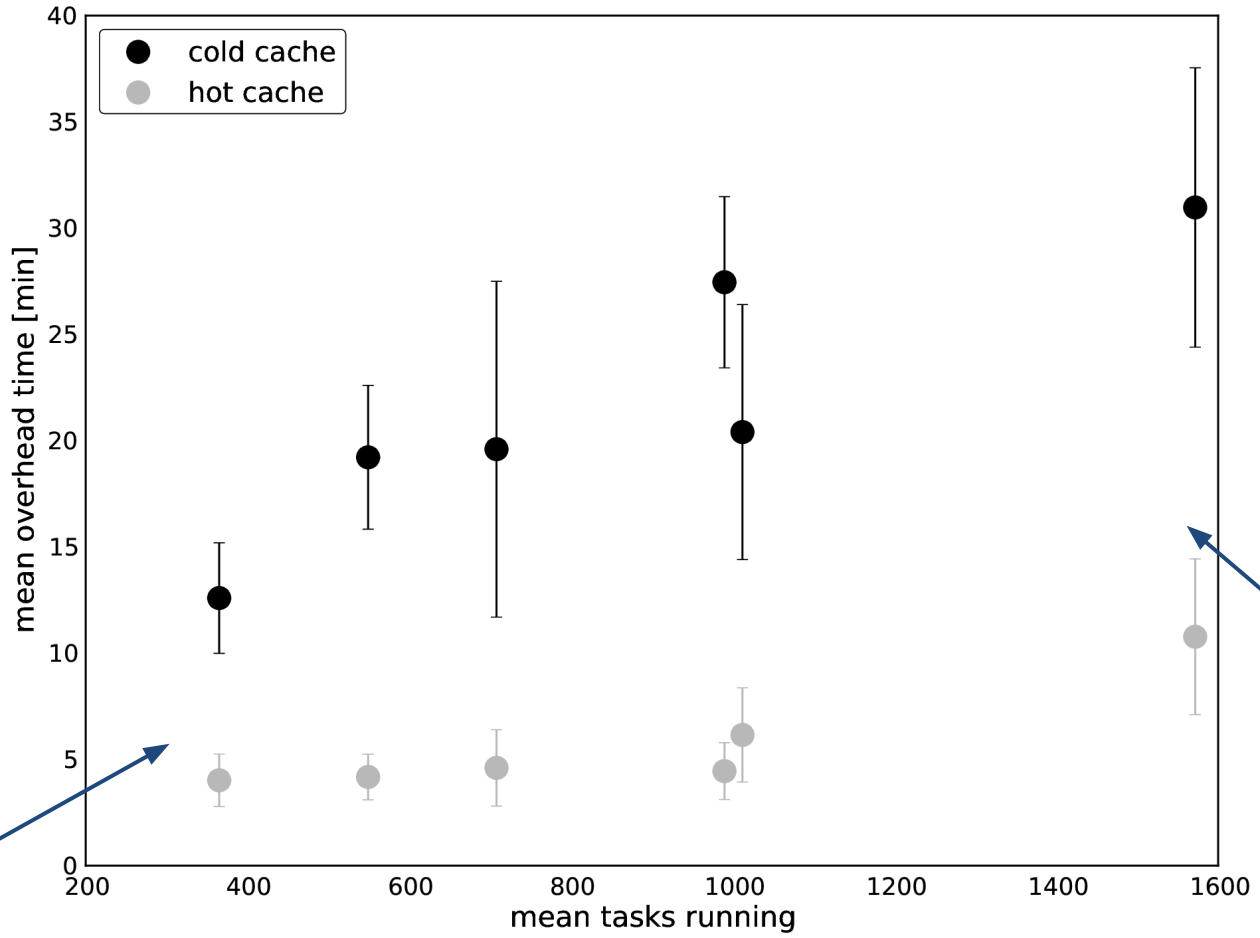
(ETXTBSY issue recently fixed!)

Measuring overheads



(a maximum of 4 tasks per worker/condor job)

Measuring overheads



few tasks, overhead mostly from parrot.

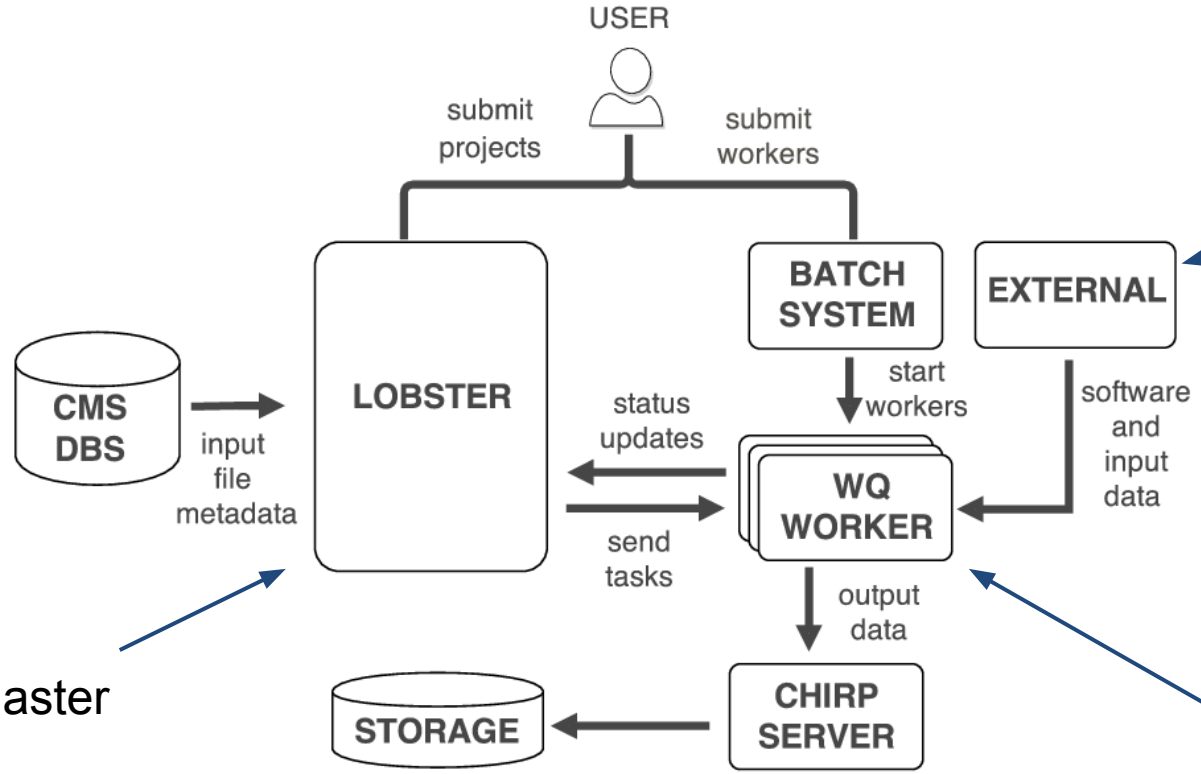
many tasks, overhead from other parts of lobster

from dedicated to opportunistic

1. How a task may access CVMFS resources?
2. How can several tasks efficiently access the same data on a node?
3. **How tasks can be sent to a computational node and managed?**
4. How should tasks be decomposed to efficiently deal with eviction?
5. How the results of several tasks should be synthesized?

How tasks can be sent to a computational node?

We use Work Queue, a master-worker lightweight execution system part of CCTools.



cvmfs via parrot

WQ master

Condor jobs are work queue workers. The Lobster master sends tasks to those workers.

The Lobster master

The master is written using Work Queue's python bindings.

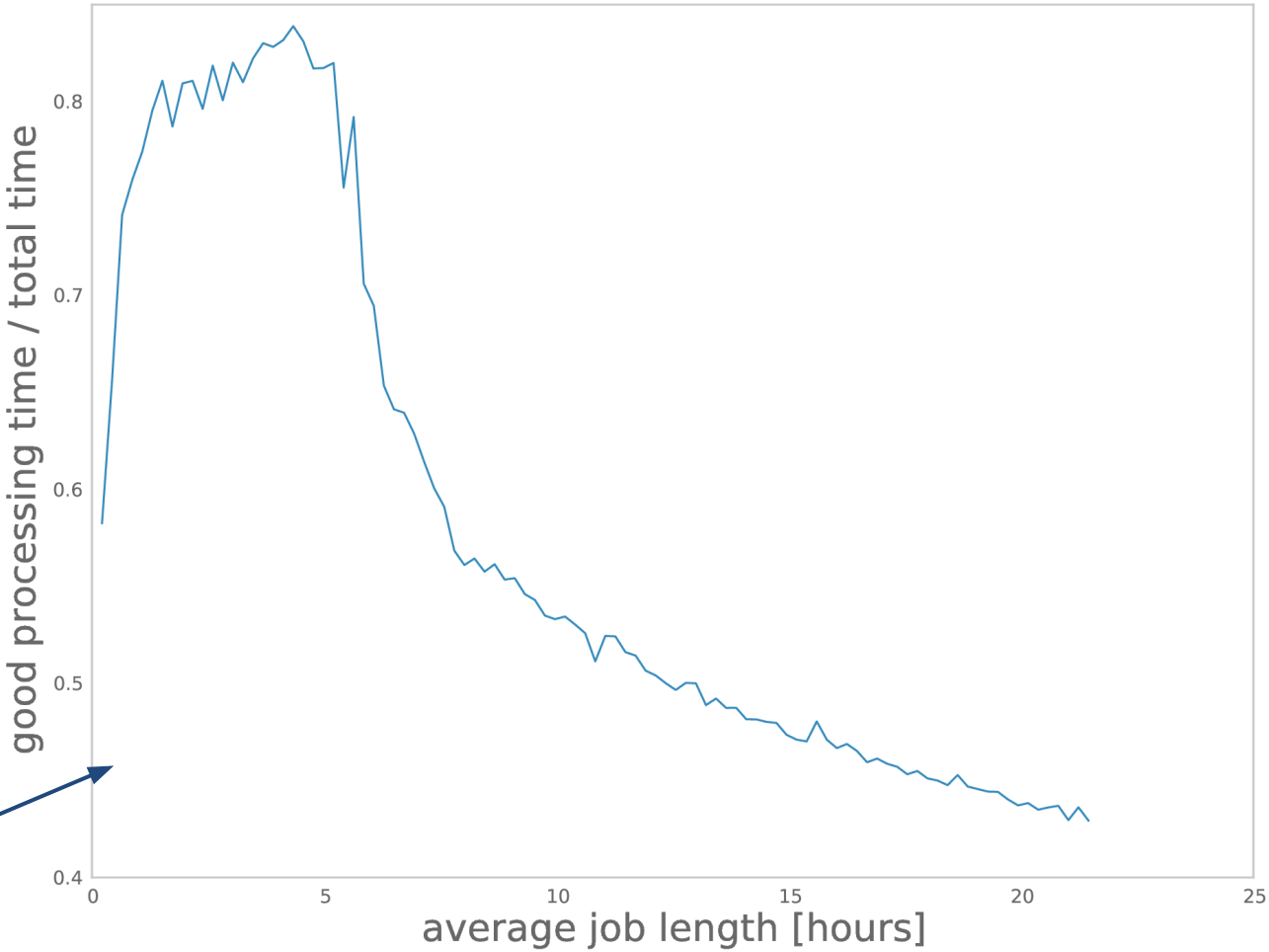
GitHub, Inc. [US] https://github.com/matz-e/lobster/blob/master/lobster/core.py

```
128
129     wq.cctools_debug_flags_set("all")
130     wq.cctools_debug_config_file(os.path.join(workdir, "work_queue_c
131     wq.cctools_debug_config_file_size(1 << 29)
132
133     queue = wq.WorkQueue(-1)
134     queue.specify_log(os.path.join(workdir, "work_queue.log"))
135     queue.specify_name("lobster_" + config["id"])
136     queue.specify_keepalive_timeout(300)
137     # queue.tune("short-timeout", 600)
138     queue.tune("transfer-outlier-factor", 4)
139     queue.specify_algorithm(wq.WORK_QUEUE_SCHEDULE_RAND)
```

from dedicated to opportunistic

1. How a task may access CVMFS resources?
2. How can several tasks efficiently access the same data on a node?
3. How tasks can be sent to a computational node and managed?
4. **How should tasks be decomposed to efficiently deal with eviction?**
5. How the results of several tasks should be synthesized?

How should tasks be decomposed to efficiently deal with eviction?

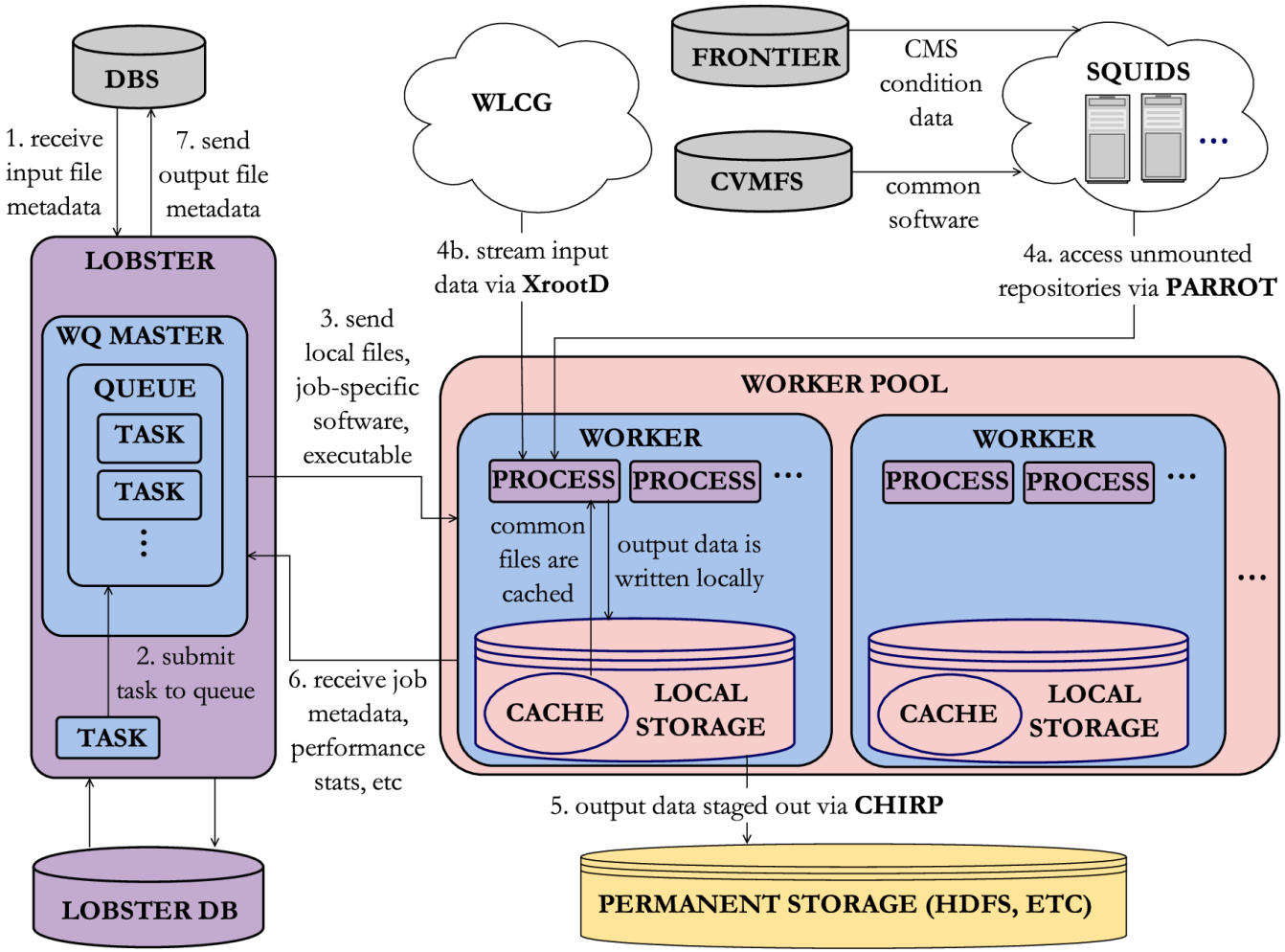


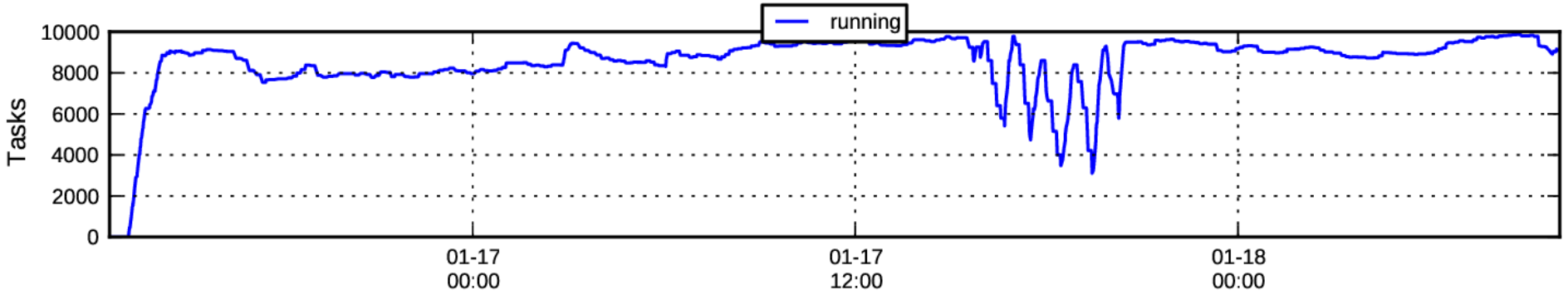
Our pool's sweet spot is 2 to 5 hours.

from dedicated to opportunistic

1. How a task may access CVMFS resources?
2. How can several tasks efficiently access the same data on a node?
3. How tasks can be sent to a computational node and managed?
4. How should tasks be decomposed to efficiently deal with eviction?
5. **How the results of several tasks should be synthesized?**

the whole lobster enchilada



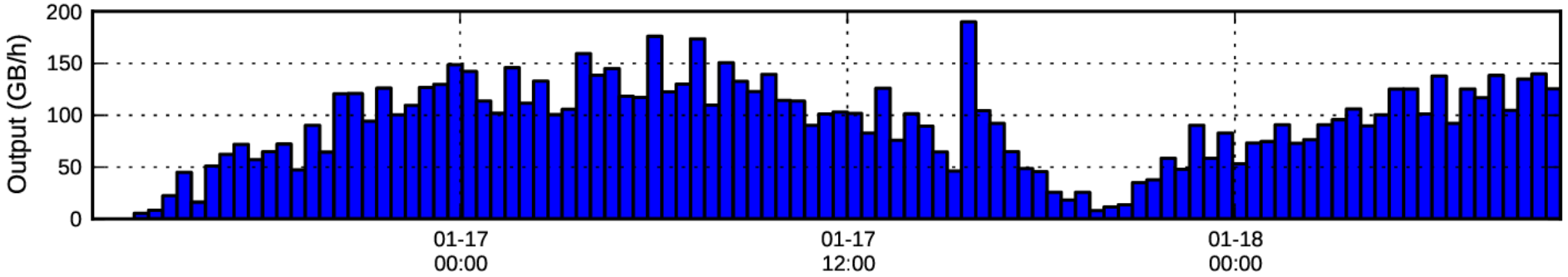


Phase	Time (hours)	Fraction (%)
Processing CPU	171036	53.4
Other Non-CPU	65356	20.4
Failed jobs	44830	14.0
WQ startup	22056	6.9
WQ Output transfer wait	8954	2.8
Total	320462	

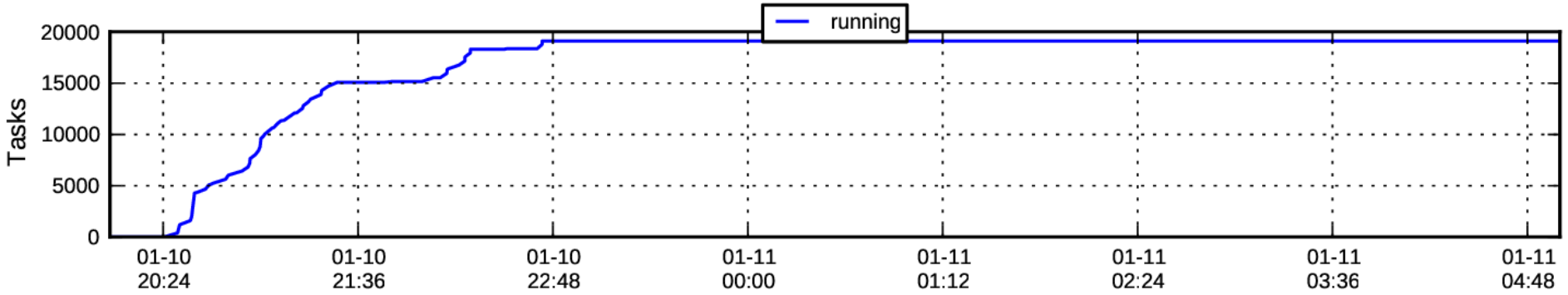
about 20 CPU years in two days.

bottlenecks

current bottleneck for O(10k) is bandwidth



next bottleneck for O(20k) is the squid proxy servers



summary

- Lobster is designed to run millions of data intensive tasks on tens of thousands of non-dedicated cores over a time scale of weeks.
- Every component runs with a minimum of privilege.
- A single user has available performance comparable to a whole Tier 2 site.
  - Running on the scale of 10k cores.
  - 9 gigabits/s input.
  - 150 gigabits/s output.

docker, condor and auto-builds

Build and Test History

i386-osx-10.6	i686-redhat6	x86_64-centos5	x86_64-centos6	x86_64-centos7	x86_64-debian8	x86_64-osx-10.9	x86_64-redhat5	x86_64-redhat6	x86_64-redhat7	x86_64-ubuntu14.04	Author	Commit
OK log tests tarball	77 / 79 log tests tarball	70 / 79 log tests tarball	73 / 79 log tests tarball	71 / 79 log tests tarball	66 / 79 log tests tarball	OK log tests tarball	OK log tests tarball	OK log tests tarball	OK log tests tarball	66 / 79 log tests tarball	Haiyan Meng	3b1fc5c8f99aac427c4b48394b40b14a986e63fb Allowing importing env variables supports both long commands and escape seque Problem: Some environment variables use backslash + newline to construct a long and some environment variables include escape sequences (e.g., PS1="\u@\h \w\$ read` command, without the `-r` option, uses backslash as an escape character, ca `PS1="\u@\h \w\$ "` to be `PS1="u@h w$ "`. the `-r` option of `read` disables th but introduces a new problem - The long commands constructed through backslas will be split into multiple line. Solution: `env -0 | while read -r -d '' line; ...`: `env - output line with NUL, not new line; -d option of `read` sets NUL as the delimite
68 / 71	78 / 81	FAIL	72 / 81	FAIL	57 / 81	68 / 71	80 / 81	80 / 81	80 / 81	57 / 81	Chao Zheng	a6aff2157e00864484d232c18736084ff8fb9e61 1. wrap task command in global wrapper command instead of writting it into shel replace sprintf by string_format.
OK	77 / 79	70 / 79	73 / 79	FAIL	66 / 79	OK	OK	OK	OK	66 / 79	Douglas Thain	bbc76315acd1b4d22fee6ca41d3416eb3c3a9712 Merge pull request #754 from dthain/master Fix WQ Tests on OS X and Linux, A
OK	77 / 79	70 / 79	73 / 79	FAIL	66 / 79	OK	OK	OK	OK	66 / 79	Douglas Thain	142ba3ba42da7b0adfecf5b9755050caeacc1afe Avoid using metric prefixes to dd, for portability.
OK log tests tarball	74 / 79 log tests tarball	67 / 79 log tests tarball	70 / 79 log tests tarball	68 / 79 log tests tarball	63 / 79 log tests tarball	OK log tests tarball	76 / 79 log tests tarball	75 / 79 log tests tarball	76 / 79 log tests tarball	63 / 79 log tests tarball	Patrick Donnelly	b432d3ec2bd22b5c36c0c9b5be4ad4b65ff11d19 Organize headers.
OK	74 / 79	67 / 79	70 / 79	FAIL	63 / 79	OK	74 / 79	76 / 79	76 / 79	63 / 79	Patrick Donnelly	e25ce3debcbeae30dd0c2b82922aa441fa1f78db Fix missing update to bytes written. This caused an assertion failure when an *at completed successfully but the return value did not indicate this. Bug found by H @hmeng-19.
OK	74 / 79	FAIL	70 / 79	FAIL	timeout	70 / 71	75 / 79	76 / 79	76 / 79	63 / 79	Patrick Donnelly	038a892a3e2b2e1feb35f809293b996ab0a530a0 Add simple assert.
OK	74 / 79	67 / 79	70 / 79	68 / 79	63 / 79	OK	76 / 79	76 / 79	76 / 79	63 / 79	Haiyan Meng	117b230ae10ccf97f603d3b84471cec189b97f43 Cloud test for cms, cms_paper and povray Shutdown the instance finally
OK	74 / 79	67 / 79	70 / 79	FAIL	63 / 79	OK	76 / 79	76 / 79	76 / 79	63 / 79	Douglas Thain	d2d4319946e6a5517cbb8eac99305e215c094040 Merge pull request #753 from dthain/master Fix WQ Tests on OSX