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
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.

Analysis software is distributed via CVMFS, a read-only filesystem over HTTP.

With FUSE, the remote software is local as far as the task is concerned.
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.
Notre Dame Condor Status

<table>
<thead>
<tr>
<th>User</th>
<th>Slots</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:dmitche6@nd.edu">dmitche6@nd.edu</a></td>
<td>4195</td>
<td>4195</td>
</tr>
<tr>
<td><a href="mailto:mzhu4@nd.edu">mzhu4@nd.edu</a></td>
<td>1764</td>
<td>1764</td>
</tr>
<tr>
<td><a href="mailto:awoodard@nd.edu">awoodard@nd.edu</a></td>
<td>89</td>
<td>356</td>
</tr>
<tr>
<td><a href="mailto:rbieler@nd.edu">rbieler@nd.edu</a></td>
<td>182</td>
<td>182</td>
</tr>
<tr>
<td><a href="mailto:apaul2@nd.edu">apaul2@nd.edu</a></td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td><a href="mailto:nblancha@nd.edu">nblancha@nd.edu</a></td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

Unclaimed: 69 slots, 319 cores
Matched: 1 slot, 8 cores
Preempting: 48 slots, 475 cores
Owner: 6405 slots, 7356 cores
Total: 6405 slots, 7356 cores

Display Options
Sort: users, machines
Show: users, states
Size: bigger, smaller
Scale: none, cores
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:

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

```python
wq.cctools_debug_flags_set("all")
wq.cctools_debug_config_file(os.path.join(workdir, "work_queue_config"))
wq.cctools_debug_config_file_size(1 << 29)

queue = wq.WorkQueue(-1)
queue.specify_log(os.path.join(workdir, "work_queue.log"))
queue.specify_name("lobster_" + config["id"])
queue.specify_keepalive_timeout(300)
# queue.tune("short-timeout", 600)
queue.tune("transfer-outlier-factor", 4)
queue.specify_algorithm(wq.WORK_QUEUE_SCHEDULE_BAND)
```
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
<table>
<thead>
<tr>
<th>Phase</th>
<th>Time (hours)</th>
<th>Fraction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing CPU</td>
<td>171036</td>
<td>53.4</td>
</tr>
<tr>
<td>Other Non-CPU</td>
<td>65356</td>
<td>20.4</td>
</tr>
<tr>
<td>Failed jobs</td>
<td>44830</td>
<td>14.0</td>
</tr>
<tr>
<td>WQ startup</td>
<td>22056</td>
<td>6.9</td>
</tr>
<tr>
<td>WQ Output transfer wait</td>
<td>8954</td>
<td>2.8</td>
</tr>
<tr>
<td>Total</td>
<td>320462</td>
<td></td>
</tr>
</tbody>
</table>

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-debian8 | x86-64-osx 10.9 | x86-64-redhat6 | x86-64-redhat7 | x86-64-ubuntu14.04 | Author          | Commit          |
|--------------|--------------|---------------|---------------|---------------|----------------|---------------|----------------|----------------|-------------------|----------------|------------------|
| **OK**       |              |               |               |               |                |               |               |                | Haiyan Meng      | 3b1fc5c3f99a427c4b43394b40b14a986e36f9 |                  |
|              | 77 / 79      | 70 / 79       | 73 / 79       | 71 / 79       | 66 / 79        | OK             | OK             | OK             |                  |                |                  |
|              | log tests    | tarball       | log tests     | log tests     | log tests      | log tests     | log tests     | log tests     |                  |                |                  |
|              |              |              |              |              |               | OK             | OK             | OK             |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  |                  |                  |
|              |              |              |              |              |               |                |               |               |                  |                  |                  |
|              |              |              |              |              |               |                |               |               |                  |                  |                  |
|              |              |              |              |              |               |                |               |               |                  |                  |                  |
| 68 / 71      | 78 / 81      | 72 / 81       | FAIL          | FAIL          | FAIL           | 57 / 81        | 68 / 71        | 80 / 81        | 80 / 81           | Chao Zheng       | a6af72157e00864a84d322e18736034f73b6e61 |
|              |              |              |              |              |               |               |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Douglas Thain    | bbb76315a0d1d4d22pee6ca41d3416eb3c3a9712 |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Douglas Thain    | 142ba3ba42da7b0adefcf5b97550caae1afe   |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Patrick Donnelly | b432d3ee3bd22b5c360c95b3e4ad365ff11d19 |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Patrick Donnelly | e25ce3d2bcbeae30dd0c7b33922a4d1fa1f8dcb |                  |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Patrick Donnelly | 038a892a3e2b1fe235f809293b9966ba0a530e0  |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Patrick Donnelly | 038a892a3e2b1fe235f809293b9966ba0a530e0  |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Douglas Thain    | d24d319946ea5517cb88ae99305e215c094a0400 |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  | Douglas Thain    | d24d319946ea5517cb88ae99305e215c094a0400 |
|              |              |              |              |              |               |                |               |               |                  |                |                  |
|              |              |              |              |              |               |                |               |               |                  |                |                  |

1. wrap task command in global wrapper command instead of writting it into shell replace sprintf by string_format.

Douglas Thain:

- Merge pull request #754 from dtian/master Fix WQ Tests on OS X and Linux, A
- Avoid using metric prefixes to dd, for portability.

Patrick Donnelly:

- Organize headers.
- 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 @meng-19.
- Add simple assert.

Haiyan Meng:

- Cloud test for cms, cms_paper and provay Shutdown the instance finally
- Merge pull request #753 from dtian/master Fix WQ Tests on OSX

Douglas Thain:

- Merge pull request #754 from dtian/master Fix WQ Tests on OS X and Linux, A