Seamless Scientific Computing from Laptops to Clouds

Prof. Douglas Thain, University of Notre Dame

http://www.nd.edu/~dthain
dthain@nd.edu
@ProfThain
The Cooperative Computing Lab

• We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.

• We *operate computer systems* on the O(10,000) cores: clusters, clouds, grids.

• We *conduct computer science* research in the context of real people and problems.

• We *develop open source software* for large scale distributed computing.

http://ccl.cse.nd.edu
The Cooperative Computing Lab

About the CCL

We design software that enables our collaborators to easily harness large scale distributed systems such as clusters, clouds, and grids. We perform fundamental computer science research in that enables new discoveries through computing in fields such as physics, chemistry, bioinformatics, biometrics, and data mining.

CCL News and Blog

- Virtual Wide Tunnel in IEEE CSE (09 Sep 2015)
- Three Papers at IEEE Cluster in Chicago (07 Sep 2015)
- CCL Language Tools 5.2.0 released (19 Aug 2015)
- Recent CCL Grad Students Take Faculty Positions (18 Aug 2015)
- CMS Analysis on 10K cores using Lobster (14 Aug 2015)
- CCL Tools 5.1.0 released (16 Jul 2015)
- CCL Tools 5.0.0 released (07 Jul 2015)
- Preservation Framework for Computational Reproducibility at ICCS 2015 (01 Jul 2015) (more news)

Community Highlight

**ForceBalance** is an open source software tool for creating accurate force fields for molecular mechanics simulation using flexible combinations of reference data from experimental measurements and theoretical calculations. These force fields are used to simulate the dynamics and physical properties of molecules in chemistry and biochemistry.

The Work Queue framework gives ForceBalance the ability to distribute computationally intensive components of a force field optimization calculation in a highly flexible way. For example, each optimization cycle launched by ForceBalance may require running 50 molecular dynamics simulations, each of which may take 10-20 hours on a high end NVIDIA GPU. While GPU computing resources are available, it is rare to find 50 available GPU nodes on any single supercomputer or HPC cluster. With Work Queue, it is possible to distribute the simulations across several HPC clusters, including the Certainty HPC cluster at Stanford, the Keeneland GPU cluster managed by Georgia Tech and Oak Ridge National Laboratories, and the Stampede supercomputer managed by the University of Texas. This makes it possible to run many simulations in parallel and complete the high level optimization in weeks instead of years.

- Lee Ping Wang, Stanford University

Research

- Papers
- Projects
- People
- Jobs
- REU

Software

- Download
- Manuals
- Makeflow
- Work Queue
- Parrot
- Chirp
- SAND
- AWF

Community

- Forum
- Getting Help
- Highlights
- Annual Meeting
- Workshops
- For Developers

Operations

- Condor Display
- Condor Pool
- Hadoop Cluster
- Biocompute
- BXGrid
- Condor Log Analyzer
- Internal

http://ccl.cse.nd.edu
Some of Our Collaborators

K. Lannon: Analyze 2PB of data produced by the LHC experiment at CERN

J. Izaguirre: Simulate 10M different configurations of a complex protein.

S. Emrich: Analyze DNA in thousands of genomes for similar subsequences.

P. Flynn: Computational experiments on millions of acquired face videos.
From the scientist's perspective...

It took a while (most of a year) but now I have my code written, installed, debugged, calibrated, and verified on my laptop.

Now I want to run at a scale 1000x larger by using a cluster, cloud, grid, or whatever you computer people are calling it today.

There is no way you are going to convince me to re-write this valuable program in order to run on your crazy cluster / OS / framework!

(Important science codes outlive OS/HW.)
On my laptop...

"sim.exe –p 50 in.dat -o output.dat"

Output!
But to run on the cloud...

"sim.exe -p 50 in.dat -o output.dat"

- **Input File**
- **Calib File**
- **Job Description File**

**Exec. Environment**

- **Unix Program**

**Software and Scripts (and OS?)**

**Resources:**
- 4 cores
- 8GB RAM
- 3GB Disk
- 1 Hour

**Submit**

**Batch System:**
- HTCondor
- Sun Grid Engine
- PBS/Torque

**Cluster Mgr:**
- Mesos
- Kubernetes
- Amazon Batch
Outline

• The Laptop Perspective
• **Scaling Up with Makeflow and Work Queue**
• VC3: Virtual Clusters
• Problem: Software Deployment
• Problem: Resource Sizing
• Lessons Learned
Makeflow = Make + Workflow

- Provides portability across batch systems.
- Enables parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.
- Transactional semantics for job execution.

Makeflow

Local  HTCondor  Torque  Work Queue  Amazon

http://ccl.cse.nd.edu/software/software/makeflow
Makeflow Shapes a Workflow

- Millions of Tasks
- Concurrency and Policy Control
- Make Flow
- Cluster or Cloud
- Precise Cleanup
- Transaction Log
- Performance Monitoring
Example: Species Distribution Modeling

Full Workflow:
12,500 species
x 15 climate scenarios
x 6 experiments
x 500 MB per projection
= 1.1M jobs, 72TB of output

Small Example: 10 species x 10 expts
More Examples

http://github.com/cooperative-computing-lab/makeflow-examples
Limitations of Direct Submission to Batch Systems

• Dispatch Latency
  – Rule of Thumb: 30 seconds to provision a resource and start a job. (UGE, HTCondor, Amazon, …)

• No Data Locality
  – Same input files get sent again and again to the same jobs. (sometimes even on the same host.)

• Solution: Deploy Work Queue Overlay
  – Accelerate subsequent job starts.
  – Share data between concurrent tasks on a node.
Work Queue Architecture

Submit → Work Queue Master

Complete

Send files

Send tasks

4-core machine

Worker Process

Local Files and Programs

A B C

Cache Dir

Task.1 Sandbox

Task.2 Sandbox

2-core task

2-core task

A

B

A

B

C

T

A

T

A

C
Harness Multiple Resources

- Work Queue Master
- Local Files and Programs
- Submit tasks
- torque_submit_workers
- condor_submit_workers
- ssh
- National Computing Resource
- Campus HTCondor Pool
- Private Cluster
- Public Cloud Provider

Thousands of Workers in a Personal Cloud
Work Queue Applications

Nanoreactor MD Simulations

Scalable Assembler at Notre Dame

ForceBalance

Adaptive Weighted Ensemble

Lobster HEP
Scaling Up Problems

• OS and Software Environment
  – User has a long and vague history of installing software packages on demand to solve problems.

• Exact Specification of Data Dependencies
  – "Oh, I forgot about that 1TB calibration file."

• Resource Selection
  – "My laptop has 16GB RAM!"
  – "But how much does the application need?"

• Time/Cost of Moving Data
  – Does it pay to move 1TB of data in order to run one task for one hour? What about 1M tasks?
Outline

• The Laptop Perspective
• Scaling Up with Makeflow and Work Queue
• VC3: Virtual Clusters
• Problem: Software Deployment
• Problem: Resource Sizing
• Lessons Learned
VC3: Virtual Clusters for Community Computation

Douglas Thain, University of Notre Dame
Rob Gardner, University of Chicago
John Hover, Brookhaven National Lab

http://virtualclusters.org
You have developed a large scale workload which runs successfully at a University cluster.

Now, you want to migrate and expand that application to national-scale infrastructure. (And allow others to easily access and run similar workloads.)

Traditional HPC Facility  Distributed HTC Facility  Commercial Cloud
Concept: Virtual Cluster

- 200 nodes of 24 cores and 64GB RAM/node
- 150GB local disk per node
- 100TB shared storage space
- 10Gb outgoing public internet access for data
- CMS software 8.1.3 and python 2.7

Virtual Cluster Service

Virtual Cluster Factory

Deploy Services

Traditional HPC Facility

Distributed HTC Facility

Commercial Cloud
Individual clusters

Distributed resources: diverse & complex

Virtual cluster

One environment: uniform & simple
Create a virtual cluster!

End user accesses the VC head node.

VC3 Architecture

User Portal

Cluster Spec

VC3 Service Instance

Middleware Scheduler

Software Catalog

Site Catalog

Batch System

MW Node

MW Node

MW Node

Resource Provider

Resource Provider

Resource Provider

MW Node

MW Node

MW Node

MW Node

MW Node
Teardown is Critical!

Destroy my virtual cluster!
Teardown is Critical!

Destroy my virtual cluster!
Outline

• The Laptop Perspective
• Scaling Up with Makeflow and Work Queue
• VC3: Virtual Clusters
• Problem: Software Deployment
• Problem: Resource Sizing
• Lessons Learned
Problem: Software Deployment

• Getting software installed on a new site is a big pain! The user (probably) knows the top level package, but doesn't know:
  – How they set up the package (sometime last year)
  – Dependencies of the top-level package.
  – Which packages are system default vs optional
  – How to import the package into their environment via PATH, LD_LIBRARY_PATH, etc.

• Many scientific codes are not distributed via rpm, yum, pkg, etc. (and user isn't root)
"I just need BLAST."
"Oh wait, I need Python!"
"Sorry, Python 2.7.12"
"Python requires SSL?"
"What on earth is pcre?"
"I give up!"
MAKER Bioinformatics Pipeline
VC3-Builder Architecture

Software Recipes

Upstream Sources

Sealed Package

Recipe

A

B

C

D

Archival or Disconnected Operation

Builder

Cached Recipes

Cached Sources

Install Tree

PATH

PYTHONPATH

LD_LIBRARY_PATH

A

B

C

D

Task Sandbox

Task
"vc3-builder –require ncbi-blast"

..Plan:  ncbi-blast => [, ]
..Try:   ncbi-blast => v2.2.28
....Plan:  perl => [v5.008, ]
....Try:   perl => v5.10.0
....could not add any source for: perl v5.010 => [v5.8.0, ]
....Try:   perl => v5.16.0
....could not add any source for: perl v5.016 => [v5.8.0, ]
....Try:   perl => v5.24.0
......Plan:  perl-vc3-modules => [v0.001.000, ]
......Try:   perl-vc3-modules => v0.1.0
......Success: perl-vc3-modules v0.1.0 => [v0.1.0, ]
....Success: perl v5.24.0 => [v5.8.0, ]
....Plan:  python => [v2.006, ]
....Try:   python => v2.6.0
....could not add any source for: python v2.006 => [v2.6.0, ]
....Try:   python => v2.7.12
......Plan: openssl => [v1.000, ]
………………
details: /tmp/test/vc3-root/x86_64/redhat6/python/v2.7.12/python-build-log
processing for ncbi-blast-v2.2.28
preparing 'ncbi-blast' for x86_64/redhat6
..Download ncbi-blast-2.2.28+-x64-linux.tar.gz' from http://download.virtualclusters.org/download/vc3/pc3_root/x86_64/redhat6/ncbi-blast/v2.2.28/ncbi-blast-build-log
details: /tmp/test/vc3-root/x86_64/redhat6/ncbi-blast/v2.2.28/ncbi-blast-build-log

(New Shell with Desired Environment)

bash$ which blastx
/tmp/test/vc3-root/x86_64/redhat6/ncbi-blast/v2.2.28/bin/blastx

bash$ blastx -help
USAGE
blastx [-h] [-help] [-import_search_strategy filename]
....

bash$ exit
Problem: Long Build on Head Node

• Many computing sites limit the amount of work that can be done on the head node, so as to maintain quality of service for everyone.

• Solution: Move the build jobs out to the cluster nodes. (Which may not have network connections.)

• Idea: Reduce the problem to something we already know how to do: Workflow!

• But how do we bootstrap the workflow software? With the builder!
vc3-builder
   --require makeflow
   --require ncbi-blast
   --
makeflow -T condor blast.mf
Bootstrapping a Workflow

Software Recipes

Upstream Sources

Builder

Makeflow

Makeflow

Build Makeflow

Build BLAST

BLAST Task

BLAST Task

BLAST Task

BLAST Task

BLAST Task

BLAST Task

BLAST Task

BLAST Task
Typically, 2-3x faster overall. But more importantly, filesystem-intensive jobs run on the cluster resources, not on the head node!
Outline

• The Laptop Perspective
• Scaling Up with Makeflow and Work Queue
• VC3: Virtual Clusters
• Problem: Software Deployment
• Problem: Resource Sizing
• Lessons Learned
Problem: Resource Selection

• Common misconceptions:
  – Users know what resources their jobs need.
    • "I need 4GB RAM, 8 cores, and 1TB disk."
  – Jobs in the same batch are usually the same.

• In reality:
  – Users have no idea what they jobs actually need!
    • "Well, it runs on my laptop, what does that have?"
  – Jobs in the same batch can have a very complex distribution of resource needs.
The Resource Sizing Problem

- 16 CPUs
- 32 GB RAM

Machine Allocation

- (External Frag)
- (Internal Frag)

Jobs
Client vs. Resource Provider

Client selects allocation:
Too big? Wasted resources.
Too small? Job fails, retry.

Provider places the allocation:
Too big? Get paid.
Too small? Still get paid!
Scheduling is not the client's problem.
"Slicing the Infinite Cake"

Allocations Too Big

Allocations Just Right: 2x Throughput
How do we know how big?

```
"command": "python task.py parameters.json",
"taskid": "195",
"user": "mfrohike",
"category": "ttZ_mAODv2",
"executable_type": "dynamic",
"monitor_version": "6.0.0.0bdd9ca9",
"exit_status": 143,
"exit_type": "limits",
"limits_exceeded": {
  "memory": [2000,"MB"]
},
"start": [1454163721,"s"],
"end": [1454165828,"s"],
"wall_time": [2107,"s"],
"cpu_time": [2066,"s"],
"cores": 2,
"cores_avg": 0.98,
"concurrent_procs": 8,
"total_procs": 273,
"virtual_memory": [2255,"MB"],
"memory": [2142,"MB"],
"swap_memory": [0,"MB"],
"bytes_read": [2274265095,"MB"],
"bytes_written": [104022016,"MB"],
"bytes_sent": [0,"MB"],
"bytes_received": [0,"MB"],
"bandwidth": [0,"Mbps"],
"total_files": 1175,
"disk": [104,"MB"]
```
Surprise: Complex Distributions!

How to pick the first allocation?

Ben Tovar says:
Minimize probability of first attempt succeeding + fallback succeeding, weighted by resources.

\[
E[\text{waste}(r, \tau, a_1)] = \int_0^\infty \left( \int_0^{a_1} (a_1 - r)\tau p(r, \tau)dr \right) d\tau + \int_0^{a_1} \left( (a_m + a_1 - r)\tau p(r, \tau)dr \right) d\tau
\]

\[
= a_1 \int_0^{a_m} \int_0^\infty \tau p(r, \tau) d\tau dr
\]

\[
+ a_m \int_0^{a_1} \int_0^{\infty} \tau p(r|\tau)d\tau p(r)dr
\]

\[
- \int_0^\infty \int_0^{\infty} r\tau p(r, \tau)d\tau dr,
\]
Production Application: Lobster

• Lobster: High energy physics analysis workload harnesses heterogeneous non-dedicated resources at Notre Dame.
• 535,078 tasks run on 25,000 core cluster over several months with the resource monitor.
• Five categories of tasks identified by user: DIGI (22911), LHEGS(500K), mAOD (2544), RECO (11582)
Resource Selection Approaches

<table>
<thead>
<tr>
<th>resource</th>
<th>naive</th>
<th>brute-force</th>
<th>min. waste</th>
<th>max. through</th>
</tr>
</thead>
<tbody>
<tr>
<td>max. peak</td>
<td>$P(0.95 &gt; r)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cores (cores)</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>cores_avg (cores)</td>
<td>2.9</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>memory (MB)</td>
<td>3830</td>
<td>2416</td>
<td>1350</td>
<td>1350</td>
</tr>
<tr>
<td>disk (MB)</td>
<td>2657</td>
<td>1338</td>
<td>1300</td>
<td>1300</td>
</tr>
</tbody>
</table>

**first allocation**

<table>
<thead>
<tr>
<th>resource</th>
<th>Equation 2</th>
<th>Equation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cores (cores)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>cores_avg (cores)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>memory (MB)</td>
<td>1350</td>
<td>1350</td>
</tr>
<tr>
<td>disk (MB)</td>
<td>1300</td>
<td>1300</td>
</tr>
</tbody>
</table>

**proportion of wasted resources per task**

<table>
<thead>
<tr>
<th>resource</th>
<th>cores</th>
<th>cores_avg</th>
<th>memory</th>
<th>disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>proportion</td>
<td>58%</td>
<td>34%</td>
<td>72%</td>
<td>55%</td>
</tr>
</tbody>
</table>

**throughput normalized**

<table>
<thead>
<tr>
<th>resource</th>
<th>cores</th>
<th>cores_avg</th>
<th>memory</th>
<th>disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>1.00</td>
<td>1.58</td>
<td>1.74</td>
<td>1.88</td>
</tr>
</tbody>
</table>

**percentage of tasks retried**

<table>
<thead>
<tr>
<th>resource</th>
<th>cores</th>
<th>cores_avg</th>
<th>memory</th>
<th>disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>percentage</td>
<td>0%</td>
<td>5%</td>
<td>7%</td>
<td>6%</td>
</tr>
</tbody>
</table>

**overhead**

| overhead (s) | 0.78 | 0.83 | 0.07 | 0.06 |

538078 tasks read in 27.60 seconds
What's the upshot?

• By selecting first allocations appropriately, we double the throughput of the system while accepting a 9 percent task failure rate.
• This approach is applied entirely from the client side, without provider assistance.
• Same approach can be applied to any cluster/cloud/grid with simple techniques.
Outline

• The Laptop Perspective
• Scaling Up with Makeflow and Work Queue
• VC3: Virtual Clusters
• Problem: Software Deployment
• Problem: Resource Sizing
• Lessons Learned
Thoughts and Lessons Learned

• Make software dependencies explicit!
  – Proposed: Nothing should be available by default, all software should require an "import" step.

• Make resource consumption more visible!
  – The laconic nature of the shell hides too much about resource consumption.

• Model observed behavior, but have a fallback when the model fails.
  – Example: first allocation based on previous behavior falls back to maximum task/machine size.
Acknowledgements

People in the Cooperative Computing Lab

Douglas Thain Director
Benjamin Tovar Research Soft. Engineer
Peter Ivie
Nicholas Hazekamp
Charles Zheng
Nathaniel Kremer-Herman
Tim Shaffer
Kyle Sweeney
James Sweet
Caitlin Guccione
Cameron Lamir
Apply Today!

DE-SC0015711
VC3: Virtual Clusters for Community Computation

ACI-1642409
SI2-SSE: Scaling up Science on Cyberinfrastructure with the Cooperative Computing Tools
About the CCL

We design software that enables our collaborators to easily harness large scale distributed systems such as clusters, clouds, and grids. We perform fundamental computer science research in that enables new discoveries through computing in fields such as physics, chemistry, bioinformatics, biometrics, and data mining.

CCL News and Blog

- Preservation Talk at iPee 2015 (02 Nov 2015)
- CMS Case Study Paper at CHEP (20 Oct 2015)
- OpenMalaria Preservation with Umbrella (19 Oct 2015)
- Virtual Wind Tunnel in IEEE CSE (09 Sep 2015)
- Three Papers at IEEE Cluster in Chicago (07 Sep 2015)
- Cctools 5.2.0 released (18 Aug 2015)
- Recent CCL Grads Take Faculty Positions (18 Aug 2015)
- (more news)

Community Highlight

Scientists searching for the Higgs boson have profited from Parrot's new support for the CoraVM Filesystem (CVMFS), a network filesystem tailored to providing world-wide access to software installations. By using Parrot, CVMFS, and additional components integrated by the Any Data, Anytime, Anywhere project, physicists working in the Compact Muon Solenoid experiment now have access to a uniform computing environment across their institution. Instead of maintaining large software collections at each participating institution, Parrot is used to maintain a centrally-hosted installation. Software files are downloaded as needed and automatically updated. A pilot project at the University of Wisconsin-Madison has demonstrated the feasibility of this approach using compute jobs to run in the Open Science Grid, harnessing 370,000 CPU-hours across 1500 compute nodes and 400 gigabytes of software in the repository.

- Dan Bradley, University of Wisconsin-Madison

http://ccl.cse.nd.edu

@ProfThain