Introduction to Makeflow and Work Queue with Containers

Nick Hazekamp and Kyle Sweeney
University of Notre Dame
nhazekam|ksweene3@nd.edu
Go to http://ccl.cse.nd.edu and Click on Container Camp Tutorial

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 that enables new discoveries through computing in fields such as physics, chemistry, bioinformatics, biometrics, and data mining.

CCL News and Blog
- Submit Your CCL Highlight (15 Jan 2018)
- CCL on Chameleon Cloud with ACIC (04 Dec 2017)
- TPDS Paper: Storage Management in Makeflow (04 Dec 2017)
- CCL at Supercomputing 2011 (13 Nov 2017)
- TPDS Paper: Job Sizing (26 Oct 2017)
- Makeflow Feature: JX Reorganization (18 Oct 2017)
- Announcement: CCTools 6.2.0 released (09 Oct 2017)
- 2017 DISC Summer REU Conclusion (09 Aug 2017)
- Announcement: CCTools 6.1.6 released (29 Aug 2017)
- (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 Carthay 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

(Submit Your Story Here)
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
First Session
- Thinking Opportunistically
- Overview of the Cooperative Computing Tools
- Makeflow
- Makeflow + Work Queue
- Hands-On Tutorial

Second Session
- Containers in Makeflow
- Hands-On Tutorial
- Work Queue API
Thinking Opportunistically
Much of scientific computing is done in conventional computing centers with a fixed operating environment with professional sysadmins.

But, there exists a large amount of computing power available to end users that is not prepared or tailored to your specific application:

- National HPC facility
- Campus-level cluster and batch system.
- Volunteer computing systems: Condor, BOINC, etc.
- Cloud services.

Can we effectively use these systems for “long tail” scientific computing?
Opportunistic Challenges

- When borrowing someone else’s machines, you cannot change the OS distribution, update RPMs, patch kernels, run as root...

- This often puts important technology just out of reach of the end user, e.g.:
  - FUSE might be installed, but without setuid binary.
  - Docker might be available, but you aren’t a member of the required Unix group.

- The resource management policies of the hosting system may work against you:
  - Preemption due to submission by higher priority users.
  - Limitations on execution time and disk space.
  - Firewalls only allow certain kinds of network connections.
Backfilling HPC with Condor at Notre Dame
Users of Opportunistic Cycles
I can get as many machines on the cloud/grid as I want!

How do I organize my application to run on those machines?
Cooperative Computing Tools
Our Philosophy

- Harness all available resources: 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...
- Make simple things easy, and complex things possible.

**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, FreeBSD, ...
- Interoperates with many distributed computing systems.
  - Condor, SGE, Torque, Globus, iRODS, Hadoop...
- Components:
  - Makeflow – A portable workflow manager.
  - Work Queue – A lightweight distributed execution system.
  - Parrot – A personal user-level virtual file system.
  - Chirp – A user-level distributed filesystem.

http://ccl.cse.nd.edu/software
### MAKEFLOW (MAKE + WORKFLOW)

- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.

[Diagram of MAKEFLOW process]

**Makeflow**

- Local
- Condor
- SGE
- Work Queue

[Link to Makeflow website]

http://ccl.cse.nd.edu/software/makeflow
#include "work_queue.h"
while( not done ) {

    while (more work ready) {
        task = work_queue_task_create();
        // add some details to the task
        work_queue_submit(queue, task);
    }

    task = work_queue_wait(queue);
    // process the completed task
}
Parrot Virtual File System

Unix Appl

Capture System
Calls via ptrace

Parrot Virtual File System

Custom Namespace

/home = /chirp/server/myhome
/software = /cvmfs/cms.cern.ch/cmssoft

File Access Tracing
Sanding Operations
User ID Mapping

Local iRODS Chirp HTTP CVMFS

http://ccl.cse.nd.edu/software/parrot
Lots of Documentation

http://ccl.cse.nd.edu
Makeflow

A Portable Workflow System
MAKЕFLOW (MAKE + WORKFLOW)

- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Trickle out work to batch system
- Fault tolerance at multiple scales.
- Data and resource management.

Makeflow

Local  Condor  SGE  Work Queue
MAKEFLOW (MAKE + WORKFLOW) 
BASED OFF AN OLD IDEA: MAKEFILES

part1 part2 part3: input.data split.py
./split.py input.data

out1: part1 mysim.exe
./mysim.exe part1 >out1

out2: part2 mysim.exe
./mysim.exe part2 >out2

out3: part3 mysim.exe
./mysim.exe part3 >out3

result: out1 out2 out3 join.py
./join.py out1 out2 out3 > result

part3 mysim.exe
part2 mysim.exe
part1 mysim.exe

split.py input.data

result
Makeflow Syntax

[output files] : [input files]

[command to run]

sim.exe
in.dat

One Rule

out.txt : in.dat calib.dat sim.exe

sim.exe in.dat –p 50 > out.txt

out.txt : in.data sim.exe

sim.exe in.data –p 50 > out.txt

Not Quite Right!
out.10 : in.dat calib.dat sim.exe
    sim.exe –p 10 in.data > out.10

out.20 : in.dat calib.dat sim.exe
    sim.exe –p 20 in.data > out.20

out.30 : in.dat calib.dat sim.exe
    sim.exe –p 30 in.data > out.30
How to run a Makeflow

• Run a workflow locally (multicore?)
  `makeflow -T local sims.mf`

• Clean up the workflow outputs:
  `makeflow –c sims.mf`

• Run the workflow on Torque:
  `makeflow –T torque sims.mf`

• Run the workflow on Condor:
  `makeflow –T condor sims.mf`
Visualization with DOT

- **makeflow_viz** –D example.mf > example.dot
- **dot** –T gif < example.dot > example.gif

DOT and related tools:
http://www.graphviz.org
Makeflow Shapes a Workflow

- Millions of Tasks
- Concurrency Control
- Thousands of Nodes
- Batch System
- T T T T
- Precise Cleanup
- Transaction Log
- Performance Monitoring
Example: Biocompute Portal

- Generate Makeflow
- Run Makeflow
- Makeflow
- Transaction Log
- Update Status
- Progress Bar
- BLAST
- SSAHA
- SHRIMP
- EST
- MAKER
- ...
Makeflow + Work Queue

A Portable Workflow System
Local Files and Programs

Makefile

Makeflow

XSEDE Torque Cluster

Private Cluster

Campus Condor Pool

Public Cloud Provider

makeflow -T torque

makeflow -T condor

???
MAKEFLOW + WORK QUEUE

Makefile → Makeflow

submit tasks

Local Files and Programs

Thousands of Workers in a Personal Cloud

XSEDE Torque Cluster

Private Cluster

Campus Condor Pool

Public Cloud Provider
Thousands of Workers in a Personal Cloud

Application API

Tasks

Master

XSede Torque Cluster

Private Cluster

Public Cloud Provider

submit tasks
Advantages of Work Queue

- Harness multiple resources simultaneously.
- Hold on to cluster nodes to execute multiple tasks rapidly.
  - (ms/task instead of min/task)
- Scale resources up and down as needed.
- Better management of data, with local caching for data intensive tasks.
- Matching of tasks to nodes with data.
To start the Makeflow

% makeflow -T wq  sims.mf

Could not create work queue on port 9123.

% makeflow -T wq -p 0 sims.mf

Listening for workers on port 8374...

To start one worker:

% work_queue_worker  master.hostname.org 8374
Start 25 Workers in Batch System

Work Queue Factory:

work_queue_factory -T slurm -w 5 -W 25

-T : specify the batch system
-w : Set the lower limit of workers to upkeep
-W : Set the upper limit of workers to submit
Keeping track of port numbers gets old fast...
Project Names

- `makeflow ... -N myproject`
- `work_queue_worker -N myproject`
- `Connect to jetstream:4057`
- `Advertise`
- `Query`
- `Catalog`
- `Query` "myproject" is at jetstream:4057
- `work_queue_status`
Project Names

Start Makeflow with a project name:

% makeflow -T wq -N myproject  sims.mf

Listening for workers on port XYZ...

Start one worker:

% work_queue_worker -N myproject

Start many workers:

% work_queue_factory -T slurm -N myproject  5
<table>
<thead>
<tr>
<th>PROJECT</th>
<th>NAME</th>
<th>PORT</th>
<th>WAITING</th>
<th>BUSY</th>
<th>COMPLETE</th>
<th>WORKERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>awe-fip35</td>
<td>fahnd04.crc.nd.edu</td>
<td>1024</td>
<td>719</td>
<td>1882</td>
<td>1206967</td>
<td>1882</td>
</tr>
<tr>
<td>hfeng-gromacs-10ps</td>
<td>lclsstor01.crc.nd.edu</td>
<td>1024</td>
<td>4980</td>
<td>0</td>
<td>1280240</td>
<td>111</td>
</tr>
<tr>
<td>hfeng2-ala5</td>
<td>lclsstor01.crc.nd.edu</td>
<td>1025</td>
<td>2404</td>
<td>140</td>
<td>1234514</td>
<td>140</td>
</tr>
<tr>
<td>forcebalance</td>
<td>leeping.Stanford.EDU</td>
<td>5817</td>
<td>1082</td>
<td>26</td>
<td>822</td>
<td>26</td>
</tr>
<tr>
<td>forcebalance</td>
<td>leeping.Stanford.EDU</td>
<td>9230</td>
<td>0</td>
<td>3</td>
<td>147</td>
<td>3</td>
</tr>
<tr>
<td>fg-tutorial</td>
<td>login1.futuregrid.tacc</td>
<td>1024</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Advantages of Work Queue

MF +WQ is fault tolerant in many different ways:

- If Makeflow crashes (or is killed) at any point, it will recover by reading the transaction log and continue where it left off.
- Makeflow keeps statistics on both network and task performance, so that excessively bad workers are avoided.
- If a worker crashes, the master detects failure and restarts the task elsewhere.
- Workers can be added and removed at any time during workflow execution.
- Multiple masters with the same project name can be added and removed while the workers remain.
- If the worker sits idle for too long (default 15m) it will exit, so as not to hold resources idle.
Alternative Makeflow Formats

Utilizing JSON/JX for easier scripting
Makeflow JSON Syntax

- Verbose flexible structure
- Familiar structure
- Consists of four items:
  - "categories": Object<Category>
  - "default_category": String
  - "environment": Object<String>
  - "rules": Array<Rule>
Makeflow JSON Syntax

```json
{
  "outputs": [{"dag_name": "out.txt"}],
  "inputs": [{"dag_name": "in.dat"}, {"dag_name": "calib.dat"},
              {"dag_name": "sim.exe"}]
  "command": "sim.exe in.dat -p 50 > out.txt",
}
```
Makeflow JSON Syntax

```json
{
    "outputs": ["out.txt"],
    "inputs": ["in.dat", "calib.dat", "sim.exe"],
    "command": "sim.exe –p 50 in.data > out.txt",
}
```
Makeflow JSON Syntax

```json
{
  "outputs": [{"dag_name": "out_10.txt"}],
  "inputs": [{"dag_name": "in.dat"}, {
    "dag_name": "calib.dat"}, {
    "dag_name": "sim.exe"}
  ],
  "command": "sim.exe –p 10 in.data > out_10.txt",
},
{
  "outputs": [{"path": "out_20.txt"}],
  "inputs": [ {
    "dag_name": "in.dat"}, {
    "dag_name": "calib.dat"}, {
    "dag_name": "sim.exe"}
  ],
  "command": "sim.exe –p 20 in.data > out_20.txt",
},...
```
Makeflow JSON Rule

- "inputs": Array<File>
- "outputs": Array<File>
- "command": String
- "local_job": Boolean
- "category": String
- "resources": Resources
- "allocation": String
- "environment": Object<String>
**Makeflow JX Syntax**

- Allows for more compact makeflows.
  - Provides functions for expanding tasks: range, variables, etc...
- Can be used as templates in conjunction with an arguments file.
- Useful for consistently structure data and different data.
```json
{
    "outputs": ["out_%d.txt"],
    "inputs": [
        "in.dat",
        "calib.dat",
        "sim.exe"
    ],
    "command": "sim.exe -p %d in.data > out_%d.txt",
} for i in range(10, 30, 10),
```
How to run a Makeflow

• Run a workflow from json
  – makeflow --json sims.json

• Clean up the workflow outputs:
  – makeflow --c --json sims.json

• Run the workflow from jx:
  – makeflow --jx sims.jx

• Run the workflow with jx and args:
  – makeflow --jx sims.jx --jx-args args.jx
Hands-on Tutorial

Short tutorial followed by Lunch
Container Integration

Providing consistent environments
Containers Create Precise Execution Environments

```
docker run ubuntu-38.23 mysim.exe
```

Ubuntu-38.23 image → Container Environment

- Code
- Libraries
- Kernel
- mysim.exe
- Kernel
Approaches to Containers with Makeflow

- **Approach 1:**
  - Create containers for starting MF and WQ, then let them run as normal.
  - You are responsible for moving container images responsibly.

- **Approach 2:**
  - Let MF create containers as needed for each task.
  - Provides more control over moving container images.
  - Sending and storing containers for each task.
Approach 1: Container for MF/WQ

- docker run ubuntu makeflow
- docker run ubuntu work_queue_worker
Approach 2: Container for Each Task

Docker Image: ubuntu-38.23

Tasks

Makeflow

Batch System

makeflow --docker ubuntu-38.23 –T sge . . .
Container Technology is Evolving

**Docker**

- `docker run ubuntu command`
- `docker.io`
- Container runs directly as a child process (still needs setuid tool, though)

**Singularity**

- `singularity exec ubuntu command`
- `singularity.lbl.gov`
- Installed service running as root
- Container runs directly as a child process (still needs setuid tool, though)
Approach 2 using Singularity

Singularity mage: ubuntu.img

Tasks

makeflow --singularity ubuntu.img –T sge . . .
Cloud Operation

Methods to Deploying
Approaches to Cloud Provisioning with Makeflow

- **Approach 1:**
  - MF creates unique instance for each task.
  - Provides complete isolation between tasks.
  - Requires startup and tear-down time of instances.

- **Approach 2:**
  - Create instances and run WQ Workers on them, submitting to WQ from MF.
  - Relies on WQ for task isolation, but caches shared files.
  - Instance management relies on the user.
Approach 1: Individual instances per task

makeflow -T amazon --amazon-config my.config ...
Approach 2: Individual instances per worker

```
work_queue_factory -T amazon --amazon-config my.config
```
Questions?

Nick Hazekamp
Email: nhazekam@nd.edu

Kyle Sweeney
Email: ksweene3@nd.edu

CCL Home: http://ccl.cse.nd.edu
Tutorial Link: http://ccl.cse.nd.edu/software/tutorials/cyversecc2018