Introduction to
Work Queue Applications

CSE 40822 – Cloud Computing – Spring 2016
Douglas Thain / Ben Tovar
Quick Recap
Makeflow = 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.

http://ccl.cse.nd.edu/software/makeflow
Makeflow + Work Queue

- Makefile
- Makeflow
- Local Files and Programs
- torque_submit_workers
- Submit tasks
- condor_submit_workers
- ssh
- Thousands of Workers in a Personal Cloud
- FutureGrid
- Torque Cluster
- Campus Condor Pool
- Private Cluster
- Public Cloud Provider
Today:
How to write Work Queue applications directly, without using Makeflow.
Makeflow vs. Work Queue

• Makeflow
  – Directed Acyclic Graph programming model.
  – Static structure known in advance.
  – All communication through files on disk.

• Work Queue
  – Submit-Wait programming model.
  – Dynamic structure decided at run-time.
  – Communicate through buffers or files.
  – More detailed knowledge of how tasks ran.
Work Queue System

X O(1000) workers running on clusters, clouds, and grids.

Master Process

Work Queue Application

submit
wait

Work Queue Library

C Python Perl

Sandbox for Each Task

worker

put sim.exe
put in.txt
exec sim.exe < in.txt > out.txt
get out.txt

in.txt sim out.txt

$ $$
Work Queue API

```c
#include "work_queue.h"

queue = work_queue_create();

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
}
```

http://ccl.cse.nd.edu/software/workqueue
Basic Queue Operations

#include “work_queue.h”
struct work_queue *queue;
struct work_queue_task *task;

// Creates a new queue listening on a port, use zero to pick any port.
queue = work_queue_create( port );
// Submits a task into a queue. (non-blocking)
work_queue_submit( queue, task );
// Waits for a task to complete, returns the complete task.
task = work_queue_wait( queue, timeout );
// Returns true if there are no tasks left in the queue.
work_queue_empty( queue );
// Returns true if the queue is hungry for more tasks.
work_queue_hungry( queue );
Basic Task Operations

#include “work_queue.h”
struct work_queue_task *task;

// Create a task that will run a given Unix command.
task = work_queue_task_create( command );

// Indicate an input or output file needed by the task.
work_queue_task Specify_file( task, name, remote_name, type, flags );

// Indicate an input buffer needed by the task.
work_queue_task Specify_buffer( task, data, length, remote_name, flags );

// Destroy the task object.
work_queue_task_delete( task );
#include “work_queue.h”

struct work_queue *queue;
struct work_queue_task *task;

queue = work_queue_create( 0 );

work_queue_specify_name( “myproject” );

task = work_queue_task_create(“sim.exe -p 50 in.dat >out.txt”);

/// Missing: Specify files needed by the task.

work_queue_submit( queue, task );

while(!work_queue_empty(queue)) {
    task = work_queue_wait( queue, 60 );
    if(task) work_queue_task_delete( task );
}
use Work_Queue;

$queue = Work_Queue->new( 0 );

$queue->specify_name( "myproject" );

$task = Work_Queue::Task->new("sim.exe -p 50 in.dat >out.txt");

### Missing: Specify files needed by the task.

$queue->submit( $task );

while(!$queue->empty()) {
    $task = $queue->wait( 60 );
    ### Missing: Do something with the task's results
}

Run One Task in Perl
from work_queue import *

queue = WorkQueue( port = 0 )

queue.specify_name( "myproject" );

task = Task("sim.exe -p 50 in.dat >out.txt")

### Missing: Specify files needed by the task.

queue.submit( task )

While not queue.empty():
    task = queue.wait(60)
C: Specify Files for a Task

```c
work_queue_taskSpecifyFile( task,"in.dat","in.dat",
    WORK_QUEUE_INPUT, WORK_QUEUE_NOCACHE );

work_queue_taskSpecifyFile( task,"calib.dat","calib.dat",
    WORK_QUEUE_INPUT, WORK_QUEUE_CACHE );

work_queue_taskSpecifyFile( task,"out.txt","out.txt",
    WORK_QUEUE_OUTPUT, WORK_QUEUE_NOCACHE );

work_queue_taskSpecifyFile( task,"sim.exe","sim.exe",
    WORK_QUEUE_INPUT, WORK_QUEUE_CACHE );
```

```
sim.exe in.dat -p 50 > out.txt
```
Perl: Specify Files for a Task

```perl
$task->specify_input_file("in.dat");
$task->specify_input_file("calib.dat");
$task->specify_output_file("out.txt");
$task->specify_input_file(local_name => "sim.exe",
                         remote_name => "sim.exe",
                         flags => $Work_Queue::WORK_QUEUE_CACHE );
```

```
sim.exe in.dat -p 50 > out.txt
```
Python: Specify Files for a Task

task.specify_file( "in.dat", "in.dat", WORK_QUEUE_INPUT, cache = False )

task.specify_input_file( "calib.dat" )

task.specify_output_file( "out.txt" )

task.specify_input_file( "sim.exe", cache = True )

sim.exe in.dat -p 50 > out.txt
You must state all the files needed by the command.
Running a Work Queue Program

gcc work_queue_example.c -o work_queue_example
-Il $HOME/cctools/include/cctools
-llwork_queue -ldttools -lm

./work_queue_example
Listening on port 8374 ...

In another window:
./work_queue_worker studentXX.cse.nd.edu 8374
setenv PYTHONPATH ${PYTHONPATH}:

${HOME}/cctools/lib/python2.6/site-package

./work_queue_example.py

Listening on port 8374 ...

In another window:

./work_queue_worker studentXX.cse.nd.edu 8374
setenv PERL5LIB \${PERL5LIB}:  (no line break)
  \${HOME}/cctools/lib/perl5/site_perl

./work_queue_example.py
Listening on port 8374 ...

In another window:
./work_queue_worker studentXX.cse.nd.edu 8374
Start Workers Everywhere

Submit workers to Condor:
condor_submit_workers studentXX.cse.nd.edu 8374 25

Submit workers to SGE:
sge_submit_workers studentXX.cse.nd.edu 8374 25

Submit workers to Torque:
torque_submit_workers studentXX.cse.nd.edu 8374 25
Work Queue: A Scalable Master/Worker Framework

Work Queue is a framework for building large master-worker applications that span many computers including clusters, clouds, and grids. Work Queue applications are written in C, Perl, or Python using a simple API that allows users to define tasks, submit them to the queue, and wait for completion. Tasks are executed by a standard worker process that can run on any available machine. Each worker calls home to the master process, arranges for data transfer, and executes the tasks. The system handles a wide variety of failures, allowing for dynamically scalable and robust applications.

Work Queue has been used to write many applications that scale up to hundreds or thousands of machines. Examples include ForceBalance, Accelerated Weighted Ensemble, the SAND genome assembly, the Makeflow workflow engine, and the AllPairs and Wavefront abstractions. The framework is easy to use, and is currently used to teach parallel and distributed programming techniques in undergraduate classes at the University of Notre Dame.

For More Information:
- Work Queue User’s Manual
- Work Queue API (C | Perl | Python)
- Work Queue Example Programs (C | Perl | Python)
- Download Work Queue
- Getting Help with Work Queue

Publications

- Badi Abdellah-Wahid, Hao-Hoa Feng, Dinesh Raja, Ronan Costas, Eric Dave, Douglas Thain, and Jesus A. Izaguirre, 
  *AWE, WQ: Fast Forwarding Molecular Dynamics using the Accelerated Weighted Ensemble*, Journal of Chemical Information and Modelling, September, 2014. DOI: 10.1021/ci500321g

- Andrew Thrasher, Zachary Musgrave, Brian Kachmark, Douglas Thain, and Scott Enrich

- Olivia Clandinin, Nicholas L. Hazekamp, Douglas Thain, Scott Enrich

- Michael Alborz, Dinesh Raja, Douglas Thain
Possible Programming Styles

- Bite off a Piece
- Run Until Convergence
- Scatter-Gather
- Heuristic Tree Search
- Hill Climbing
More Advanced Features
Use Project Names

work_queue_worker
  -N myproject

- Work Queue (port 9037)
- Worker
- Catalog

connect to studentXX:9037
advertise
query
query

work_queue_status

“myproject” is at studentXX:9037
Specify Project Names in Work Queue

Specify Project Name for Work Queue master:

**Python:**

```python
q.specify_name ("myproject")
```

**Perl:**

```perl
work_queue_specify_name ($q, "myproject");
```

**C:**

```c
work_queue_specify_name (q, "myproject");
```
Start Workers with Project Names

Start one worker:

$ work_queue_worker  -N myproject

Start many workers:

$ sge_submit_workers  -N myproject  5
$ condor_submit_workers  -N myproject  5
$ torque_submit_workers  -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>

% ./work_queue_status
Tag a Work Queue task

• You can specify a tag for a task
  – Batch together related tasks with a tag
  – Add a unique identifier as tag to quickly identify on completed.

Python:

```python
t.specify_tag("iteration_1")
```

Perl:

```perl
work_queue_task_specify_tag($t, "iteration_1");
```

C:

```c
work_queue_task_specify_tag(t, "iteration_1");
```
Cancel Work Queue task

• You can cancel any task anytime after it has been submitted to Work Queue as follows:

Python:
```python
cancel_by_taskid(q, cancel_taskid)
cancel_by_tasktag(q, "redundant_task")
```

Perl:
```perl
work_queue_cancel_by_taskid($q, $cancel_taskid);
work_queue_cancel_by_tasktag($q, "redundant_task");
```

C:
```c
work_queue_cancel_by_taskid(q, cancel_taskid);
work_queue_cancel_by_tasktag(q, "redundant_task");
```
Retry “slow” Work Queue tasks

• Running on large number of resources
  – High probability of stragglers
  – These stragglers slow down completion of your computation

• Work Queue has a “fast abort” feature to detect stragglers and migrate tasks to other available resources.
  – It keeps running average of the task execution times.
  – Activate fast abort by specifying a multiple of the average task execution time
  – Work Queue will abort any task executing beyond the specified multiple of the task execution time.
  – The aborted task will then be retried for execution on another available machine.
Activating fast abort in Work Queue

Python:

```python
# abort if task exceeds 2.5 * avg execution time
q.activate_fast_abort (2.5)
```

Perl:

```perl
work_queue_activate_fast_abort ($q, 2.5);
```

C:

```c
work_queue_activate_fast_abort (q, 2.5);
```
Send intermediate buffer data as input file for Work Queue Task

• You may have data stored in a buffer (e.g., output of a computation) that will serve as input for a work queue task.

• To send buffer data as input to a task:

Python:

```python
buffer = “This is intermediate data”
t.specify_buffer(buffer, input.txt, WORK_QUEUE_INPUT, cache=True)
```
Send intermediate buffer data as input file for Work Queue Task

Perl:
my $buffer = "This is intermediate buffer data"
my $buff_len = length($buffer);
work_queue_task Specify buffer($t,$buffer,$buf_len, "input.txt", $WORK_QUEUE_CACHE);

C:
const char * buffer = "This is intermediate buffer data";
int buff_len = strlen(buffer);
work_queue_task Specify buffer(t, buffer, buff_len, "input.txt", WORK_QUEUE_CACHE);
Work Queue Task Structure

• The Work Queue task structure contains information about the task and its execution.
• The following information is available in the task structure:
  • Task command (command line arguments)
  • Task output (stdout)
  • Task return status (exit code of command)
  • Task host (hostname and port on which it ran)
  • Task submit time, Task completion time
  • Task execution time (command)
  • And more..
Accessing Work Queue Task structure

Python:

```python
# After 't' has been retrieved through q.wait()
print % t.output
print % t.host
```

Perl:

```perl
# After 't' has been retrieved through work_queue_wait()
print "$t->{output}\n";
print "$t->{host})\n":
```

C:

```c
// After 't' has been retrieved through work_queue_wait()
printf ("%s\n", t->output);
printf ("%s\n", t->host):
```
Work Queue Statistics

• Work Queue collects statistics on tasks & workers during run time
• The statistics can be retrieved anytime during run time
• **Global statistics**
  – Total bytes sent, Total send time
  – Total bytes received, Total receive time
  – Total tasks dispatched, Total tasks complete
  – Total workers joined, Total workers removed
• **Statistics on Tasks**
  – Tasks running, Tasks complete, Tasks waiting
• **Statistics on Workers**
  – Workers ready, workers busy, workers cancelling
• And more…
Accessing Work Queue Statistics

- You can access the Work Queue statistics anytime during run time as follows:

**Python:**

```python
print q.stats
print q.stats.tasks_running
```

**Perl:**

```perl
my $work_queue_stats = work_queue_stats->new();
work_queue_get_stats ($q, $work_queue_stats);
print "$work_queue_stats->{tasks_running}";
```
Work Queue Statistics

• You can access the Work Queue statistics anytime during run time as follows:

C:

```c
struct work_queue_stats *wq_stats;
work_queue_get_stats (q, wq_stats);
printf ("%d", work_queue_stats->tasks_running);
```
Thinking Even BIGGER
Managing Your Workforce

Master A

Master B

Master C

Condor Pool

SGE Cluster

work_queue_pool –T condor 200

WQ Pool

WQ Pool

WQ Pool

Work Queue Pool


work_queue_pool –T sge 100

200

100
Multi-Slot Workers

Master

Worker

work_queue_worker
(implies 1 task, 1 core)

4 cores
512 MB

specify_cores(4);
specify_memory(512);

Worker

work_queue_worker
--cores 8
--memory 1024
Using Foremen

Approx X1000 at each fanout.

work_queue_worker
--foreman $MASTER $PORT

California

Chicago
Sample Applications of Work Queue
Replica Exchange

Simplified Algorithm:
- Submit N short simulations at different temps.
- Wait for all to complete.
- Select two simulations to swap.
- Continue all of the simulations.

Dinesh Rajan, Anthony Canino, Jesus A Izaguirre, and Douglas Thain,
Converting A High Performance Application to an Elastic Cloud Application, Cloud Com 2011
Using WQ, we could assemble a human genome in 2.5 hours on a collection of clusters, clouds, and grids with a speedup of 952X.

Christopher Moretti, Andrew Thrasher, Li Yu, Michael Olson, Scott Emrich, and Douglas Thain, A Framework for Scalable Genome Assembly on Clusters, Clouds, and Grids, IEEE Transactions on Parallel and Distributed Systems, 2012
Adaptive Weighted Ensemble

Proteins fold into a number of distinctive states, each of which affects its function in the organism.

How common is each state?
How does the protein transition between states?
How common are those transitions?
AWE on Clusters, Clouds, and Grids
Work Queue provides the **Submit-Wait** programming model using a **Master-Worker** architecture.