Creating Custom Work Queue Applications

Nicholas Hazekamp
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.
#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
Work Queue Applications

Nanoreactor MD Simulations

Scalable Assembler at Notre Dame

Lobster HEP

ForceBalance

Adaptive Weighted Ensemble
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,
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 Architecture

Application

Submit Task1(A,B)
Submit Task2(A,C)

Wait

Send files

Worker Process

Send tasks

4-core machine

Task 1
Task 2

Sandbox
2-core task

Sandbox
2-core task

Submit Task1(A,B)
Submit Task2(A,C)

Send files

Worker Process

Cache Dir

Task 1
Task 2

CCTools

University of Notre Dame
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_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 );
}

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

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

work_queue_task_specify_file(task,"calib.dat","calib.dat",
    WORK_QUEUE_INPUT, WORK_QUEUE_NOCACHE );

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

work_queue_task_specify_file( 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

```
work_queue_task_specify_file( $task,"in.dat","in.dat",
    $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file($task,"calib.dat","calib.dat",
    $WORK_QUEUE_INPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,"out.txt","out.txt",
    $WORK_QUEUE_OUTPUT, $WORK_QUEUE_NOCACHE );

work_queue_task_specify_file( $task,"sim.exe","sim.exe",
    $WORK_QUEUE_INPUT, $WORK_QUEUE_CACHE );

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

```python
# Specify files for a task

# Input files
task.specify_file("in.dat", "in.dat", WORK_QUEUE_INPUT, cache=False)
task.specify_file("calib.dat", "calib.dat", WORK_QUEUE_INPUT, cache=False)

# Output files
task.specify_file("out.txt", "out.txt", WORK_QUEUE_OUTPUT, cache=False)

task.specify_file("sim.exe", "sim.exe", WORK_QUEUE_INPUT, 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
   -I $HOME/cctools/include/cctools
   -L $HOME/cctools/lib
   -lwork_queue -ldttools  -lm

./work_queue_example
Listening on port 8374 …

In another window:
./work_queue_worker    master.host.name.org  8374
setenv PERL5LIB ${PERL5LIB}:
    ${HOME}/cctools/lib/perl5/site_perl

./work_queue_example.pl
Listening on port 8374 ...

In another window:
./work_queue_queue_worker  master.host.name.org  8374
setenv PYTHONPATH \${PYTHONPATH}: \${HOME}/cctools/lib/python2.6/site-package

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

In another window:
./work_queue_queue_worker master.host.name.org 8374
Start Workers Everywhere

Submit workers to Condor:
condor_submit_workers master.hostname.org 8374 25

Submit workers to SGE:
sge_submit_workers master.hostname.org 8374 25

Submit workers to Torque:
torque_submit_workers master.hostname.org 8374 25
Use Project Names

```
work_queue_worker -N myproject
```

---

**Diagram:**

- **Work Queue (port 9037)**
  - Connect to *india:9037*
  - Advertise

- **Worker**
  - Query

- **Catalog**
  - Query
  - Advertise
  - "myproject" is at *india:9037*
Specify Project Names in Work Queue

Specify Project Name for Work Queue master:

C:

```c
work_queue_specify_name (q, "myproject");
```

Perl:

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

Python:

```python
q.specify_name ("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
Advanced Features (in the docs)

• Submit / remove tasks by tag / name.
• Auto reschedule tasks that take too long.
• Send in-memory data as a file.
• Log and graph system performance
• Much more!
Managing Your Workforce

- Master A
- Master B
- Master C

Condor Pool
- W
- W
- W

SGE Cluster
- W
- W
- W

Condor Pool
- W
- W

work_queue_factory –T condor 200

WQ Pool

submits new workers.
restarts failed workers.
removes unneeded workers.

work_queue_factory –T sge 100
Using Foremen

Approx X1000 at each fanout.

work_queue_worker --foreman $MASTER $PORT

California

Chicago
Multi-Slot Workers

Master

Worker

specify_cores(4);
specify_memory(512);

work_queue_worker (implies 1 task, 1 core)

Worker

work_queue_worker
--cores 8
--memory 1024