Fine Grained Resource Management for Functions in Parsl and Work Queue

Douglas Thain
University of Notre Dame
3 October 2019
Parsl + Work Queue for Scalable Apps

http://parsl-project.org

http://ccl.cse.nd.edu

Work Queue: A Scalable Master/Worker Framework

Work Queue is a framework for building large master-worker applications that span thousands of machines drawn from 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 applications that scale from a handful of workstations up to tens of thousands of cores running on supercomputers. Examples include Lobster, NanoReactors, ForceBalance, Accelerated Weighted Ensemble, the SAND genome assembler, the Makerflow workflow engine, and the All-Pairs and Wavefront abstractions. The framework is easy to use, and has been used to teach courses in parallel computing, cloud computing, distributed computing, and cyberinfrastructure at the University of Notre Dame, the University of Arizona, and the University of Wisconsin - Eau Claire.

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

Powerful Pythonic Workflow Programming Model

Scalable, Portable, Robust Distributed Execution
Some Work Queue Applications

Nanoreactors
ab-initio Chemistry

ForceBalance
FF Optimization

Lobster
CMS Data Analysis

Adaptive Weighted Ensemble
Molecular Dynamics

Low-Level API:

```python
task = create(details);
submit(task);
task = wait(timeout);
```

Peter Bui, Dinesh Rajan, Badi Abdul-Wahid, Jesus Izaguirre, Douglas Thain,
Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications,
Workshop on Python for High Performance and Scientific Computing (PyHPC) at Supercomputing 2011.
**Elastic.** Workers can be added and removed during runtime, and the manager automatically uses the workers available.

**Robust.** Tasks running on workers that fail are automatically detected and handled elsewhere.

**Data Management.** Files may be cached at the workers, which reduces transfer times and network utilization. (No shared FS)

**Resource Management.** Resources such as core, memory, and disk are tracked and limited, so both tasks and workers can be heterogeneous.

**Language Agnostic.** Workers may run in campus cluster, national labs, or commercial cloud facilities. Managers can be written in Python, Perl, or C. (SWIG/JSON bindings for more)
So What's New?

Last Year: 2019

This Year: 2020

$ conda install -y -c conda-forge cctools parsl
System Architecture

Generate Funcs
Order Functions (Futures)
Functions -> Tasks
Schedule Tasks

Parsl Python Interface
Parsl Data Flow Kernel
Parsl / WQ Executor
Work Queue Manager

Local Filesystem

Thousands of Workers on National Cyberinfrastructure
HTCondor, PBS, SLURM, Amazon, Blue Waters, OSG, XSEDE...

Execute Tasks Remotely on Local Disk
Evolution of Batch Computing

"Classic" Batch Computing: One Process per Node

- 1 core
- 1 GB RAM
- Task

- 1 core
- 1 GB RAM
- Task

- 4 core

How many tasks can I run on this node at once?

What is the best node to run this program on?

Shared Filesystem

Manycore Cluster Computing: Multiple (Small) Functions per Node

- 256 cores
- 64 GB RAM
- SSD
- λ
- Task
- Task
- Task

- 256 cores

How many tasks can I run on this node at once?

Load?

Shared Filesystem
Pop Quiz!

What are the two most terrifying words in the Python language?

import tensorflow
Challenge 1: Transporting Python Environments

Direct Access

Local Unpack

Graphs showing Total Core-Hours vs. Nodes for Direct Access and Local Unpack.
Challenge 2: How many functions per node?

We must be able to measure a single function call!

LFM - Lightweight Function Monitor

funcA -> { 0.5 cores, 2.3GB RAM, 15s }
funcB -> { 2.1 cores, 0.9GB RAM, 9s }
Lightweight Function Monitors (LFMs)

Activate LFMs with an import and the `@monitored` keyword

```python
In [7]: from resource_monitor import monitored
   : from time import sleep

In [12]: # declare a function to be monitored with the `@monitored()` decorator
   : @monitored()
   : def my_function_1(wait_for):
   :     sleep(wait_for)
   :     return 'waited for {} seconds'.format(wait_for)

(result, resources) = my_function_1(0.1)
print(result, '{}').format(resources['memory'], 'wall_time': resources['wall_time'])

waited for 0.1 seconds {'memory': 49, 'wall_time': 101689}
```
Putting it All Together

Parsl Python Interface

Parsl Data Flow Kernel

Parsl / WQ Executor

Work Queue Manager

SSD

LFM

SSD

SSD
Fine-grained Management of Resources with WorkQueue

To utilize Work Queue with Parsl, please install the full CCTools software package within an appropriate Anaconda or Miniconda environment (instructions for installing Miniconda can be found here):

```
$ conda create --y --name <environment> python=<version> conda-pack
$ conda activate <environment>
$ conda install --y -c conda-forge cctools parsl
```

This creates a Conda environment on your machine with all the necessary tools and setup needed to utilize Work Queue with the Parsl library.

The following snippet shows an example configuration for using the Work Queue distributed framework to run applications on remote machines at large. This examples uses the `WorkQueueExecutor` to schedule tasks locally, and assumes that Work Queue workers have been externally connected to the master using the `work_queue_factory` or `condor_submit_workers` command line utilities from CCTools. For more information on using Work Queue or to get help with running applications using CCTools, visit the CCTools documentation online.

```python
from parsl.config import Config
from parsl.executors import WorkQueueExecutor
cfg = Config(
    executors=[
        WorkQueueExecutor(
            label="wqex_local",
            port=58055,
            project_name="WorkQueue Example",
            shared_fs=True,
            see_worker_output=True)
    ]
)
```
Contributors

U. of Notre Dame: Tim Shaffer, TJ Dasso, Andrew Litteken, Tanner Juedeman, Ben Tovar

U. Chicago: Zhuozhao Li, Yadu Babuji, Ben Clifford, Anna Woodard, Kyle Chard, Ian Foster

NSF Grant #ACI-1642409 (SI2-CCTools)
NSF Grant #OAC-1931387 (CSSI Dataswarm)
DOE SGCSR Fellowship (Shaffer)

http://ccl.cse.nd.edu

http://parsl-project.org

Work Queue