TopCoffea with the work queue executor

Ben Tovar
btovar@nd.edu
where we are

"This demo topcoffea runs on my laptop, but I need much more for the real application. It would be great if we can run $O(10K)$ tasks like this on this cloud/grid/cluster I have heard so much about."
where we want to be

Cooperative Computing Lab Tools (CCTools)
(work queue, resource monitor, ...)

CCTools Objectives

• Harness all the resources that are available: 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.
CCTools

- Open source, GNU General Public License.
- Runs on Linux, MacOS*
- Interoperates with many distributed computing systems.
  - Condor, SGE, Torque, Globus, iRODS, Hadoop...
most used components

**Makeflow**: A portable workflow manager
What to run?

**Work Queue**: A lightweight distributed execution system
What to run and where to run it?

**Chirp**: A user-level distributed filesystem
Where to get/put the data?

**Parrot**: A personal user-level virtual file system
How to read/write the data?
topcoffea as a manager-worker application
manager-worker application

all events
chunk
chunk
chunk

topcoffe
chunk
chunk
chunk

chunk = task
manager-worker application

topcoffea

In a manager worker application...
manager-worker application

The manager process generates tasks, puts them in a queue...
manager-worker application

... delivers them to worker processes to execute...
manager-worker application

... waits for workers to execute tasks ...
manager-worker application

and gathers the results on completion.
manager-worker application

and on and on until no more tasks are generated.
manager-worker application

topcoffe

worker process in campus condor cluster

worker process in Amazon cloud resources

$$$

15
TopCoffea without work queue
Need to replicate environment at workers

local machine

remote work queue worker

python environment

topcoffea + wq

topcoffea task
using the work queue executor

```python
# topcoffea/analysis/topEFT/work_queue_run.py
...
# minimum executor arguments:

executors_args = {
    'schema': NanoAODSchema,
    'master-name': '{}-wq-coffea'.format(os.environ['USER']),
    'environment-file': topeftenv.get_environment(),
    'port': 9123,  # or a range [9123, 9130]
    'extra-input-files': ['topeft.py'],
}
...
'environment-file': topeftenv.get_environment(),

- get_environment() generates a python environment that can easily be transferred to workers.
- the environment is created once per the latest git commit in the topcoffea directory.
- if there are unstaged changes, the environment is regenerated everytime!
  - (this is expensive, so remember to commit your changes)
- last three environments are kept in: topcoffea/analysis/topEFT/envs
running work queue

$ conda activate topcoffea-env
$ cd topcoffea/analysis/topEFT
$ python work_queue_run.py --chunksize 10000
..../topcoffea/cfg/mycfg.cfg

# in some other terminal, launch a worker for that manager
# workers don't need PYTHONPATH set.
# -M my-manager-name to serve managers with that name
# it could be a regexp.

# --single-shot to terminate after serving one manager
# In general workers may serve many managers in their
# lifetime, but only one at a time.
$ conda activate topcoffea-env
$ work_queue_worker -M $USER-wq-topcoffea --single-shot
how do workers find the manager?

- my name is...
  - I am at...

- catalog server
  - ccl.cse.nd.edu

- worker process

- where is a manager with name ...?
<table>
<thead>
<tr>
<th>PROJECT</th>
<th>HOST</th>
<th>PORT</th>
<th>WAITING</th>
<th>RUNNING</th>
<th>COMPLETE</th>
<th>WORKERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>geomTRIC</td>
<td>128.120.146.3</td>
<td>9000</td>
<td>823</td>
<td>0</td>
<td>761</td>
<td>0</td>
</tr>
<tr>
<td>geomTRIC</td>
<td>128.120.146.3</td>
<td>8999</td>
<td>793</td>
<td>0</td>
<td>791</td>
<td>0</td>
</tr>
<tr>
<td>btovar-wq-topcoffee</td>
<td>earth.crc.nd.edu</td>
<td>9123</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lobster_kmohrman_EFT</td>
<td>earth.crc.nd.edu</td>
<td>9000</td>
<td>2680</td>
<td>899</td>
<td>125466</td>
<td>565</td>
</tr>
<tr>
<td>lobster_rgoldouz_L1A</td>
<td>earth.crc.nd.edu</td>
<td>9001</td>
<td>0</td>
<td>7</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20003</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20007</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT_LARGE-cc5858845</td>
<td>vm65-195.iplantcollaborator</td>
<td>20015</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20001</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20012</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20008</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEMT-cc58588456b17c2</td>
<td>vm65-195.iplantcollaborator</td>
<td>20000</td>
<td>365</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

chosen manager name
create a worker in condor

# using \ to break the command in multiple lines
# you can omit the \ and put everything in one line

# run 3 workers in condor, each of size 1 cores, 2048 MB
# of memory and 4096 MB of disk,
# to serve ${USER}-my-makeflow
# and which timeout after 60s of being idle.

$ condor_submit_worker --cores 1 \ 
  --memory 2000 \ 
  --disk 4000 \ 
  -M my-manager-name \ 
  --timeout 60 \ 
  3
work queue
resource management
resources contract: running several tasks in a worker concurrently

Worker has available:

i cores
j MB of memory
k MB of disk

Task needs:

m cores
n MB of memory
o MB of disk

Task runs only if it fits in the currently available worker resources.
resources contract example

Worker has available:

8 cores  
512 MB of memory  
512 MB of disk

Task a:

4 cores  
100 MB of memory  
100 MB of disk

Task b:

3 cores  
100 MB of memory  
100 MB of disk

Tasks a and b may run in worker at the same time.  
(Work could still run another 1 core task.)
Beware!
tasks use all worker on missing declarations

Worker has available:

8 cores
512 MB of memory
500 TB of disk

Task a:
4 cores
100 MB of memory

Task b:
3 cores
100 MB of memory

Tasks a and b may NOT run in worker at the same time.
(disk resource is not specified.)
specifying tasks resources

# categories are groups of tasks with the same resource requirements

# specify resources in executor args:

executors_args = {
    'schema': NanoAODSchema,
    'master-name': '{}-wq-coffea'.format(os.environ['USER']),
    'environment-file': topeftenv.get_environment(),
    'port': 9123,  # or a range [9123, 9130]
    'extra-input-files': ["topeft.py"],
    'cores': 1,
    'memory': 2000,  # in MB
    'disk': 4000,  # in MB
}
managing resources

Do nothing (default if tasks don't declare cores, memory or disk):

One task per worker, task occupies the whole worker.

Honor contract (default if tasks declare resources):

Task declares cores, memory, and disk (the three of them!)
Worker runs as many concurrent tasks as they fit.
Tasks may use more resources than declared.

Monitoring and Enforcement:

Tasks fail (permanently) if they go above the resources declared.

Automatic resource labeling:

Tasks are retried with resources that maximize throughput, or minimize waste.
Monitoring resource usage

```python
#
executors_args = {
    'schema': NanoAODSchema,
    'master-name': '{}-wq-coffea'.format(os.environ['USER']),
    'environment-file': topeftenv.get_environment(),
    'port': 9123,  # or a range [9123, 9130]
    'extra-input-files': ['topeft.py'],
    'cores': 1,
    'memory': 2000,  # in MB
    'disk': 4000,  # in MB
    'resource-monitor': True,  # does not work in OSX
    'verbose': True
}
```
Monitoring resource usage

Task (id #1) complete: ./certs_wrapper.sh --environment full_env_d7122b27_HEAD.tar.gz --unp nction.p item_0.p output_0.p (Return code 0)
Allocated cores: 1, memory: 2000 MB, disk: 4000 MB, gpus: 0
Measured cores: 1, memory: 631 MB, disk 408 MB, gpus: 0, runtime 39.550364
Tasks fail permanently on resource exhaustion

Task (id #1) complete: ./certs_wrapper.sh --environment full_env_d7122b27_HEAD.tar.gz --unpack-to
ction.p item_0.p output_0.p (return code 143)
Allocated cores: 1, memory: 2 MB, disk: 2 MB, gpus: 0
Measured cores: 1, memory: 1 MB, disk 391 MB, gpus: 0, runtime 0.003314
Task id #1 failed with code: 16
automatic resource labeling
when you don't know how big your tasks are

Tasks which size (e.g., cores, memory, and disk) is not known until runtime.

One task per worker:
Wasted resources, reduced throughput.

Many tasks per worker:
Resource contention/exhaustion, reduced throughput
Task-in-the-Box

workers
Task-in-the-Box

Allocations inside a worker

workers
Task-in-the-Box

One task per allocation

One task per allocation

workers
Task-in-the-Box

One task per allocation

Task exhausted its allocation

workers
Task-in-the-Box

- One task per allocation
- Retry allocating a whole worker

workers
automatic resource labeling

```python
#
executors_args = {
    'schema': NanoAODSchema,
    'master-name': '{}-wq-coffea'.format(os.environ['USER']),
    'environment-file': topeftenv.get_environment(),
    'port': 9123,  # or a range [9123, 9130]
    'extra-input-files': ['topeft.py'],
    'cores': 2,  # now resources are maximum allowed
    'memory': 4000,  # in MB
    'disk': 4000,  # in MB
    'resource-monitor': True,
    'resources-mode': 'auto',
    'verbose': True
}
```
what work queue does behind the scenes

1. Some tasks are run using full workers.
2. Statistics are collected.
3. Allocations computed to maximize throughput, or minimize waste.
   a. Run task using guessed size.
   b. If task exhausts guessed size, keep retrying on full (bigger) workers, or a specified cores, memory or disk is reached.
4. When statistics become out-of-date, go to 1.
At least one task that is now waiting, failed exhausting these much of the resource.

~ no fixed resource set, and all tasks have run under this value
other
work queue
capabilities
how many workers do I submit?

my name is XYZ
I am at HOSTPORT
catalog server
  ccl.cse.nd.edu
worker process
where is a manager with name XYZ?
topcoffea
the work queue factory

my name is XYZ
I am at HOSTPORT

catalog server
ccl.cse.nd.edu

how many workers XYZ needs?

where is a manager with name XYZ?

worker process

batch system

submit
work queue workers

topcoffe

work queue factory
the work queue factory

Factory creates workers as needed by the manager:

```
$ work_queue_factory -Tcondor \
  -M ${USER}-wq-topcoffeea
  --min-workers 5
  --max-workers 200
  --cores 1 --memory 4096 --disk 10000
```
the work queue factory -- conf file

to make adjustments the configuration file can be modified once the factory is running

$ work_queue_factory -Tcondor -C my-conf.json
$ cat my-conf.json
{
   "manager-name": "btovar-wq-topcoffea",
   "max-workers": 200,
   "min-workers": 5,
   "workers-per-cycle": 5,
   "cores": 4,
   "disk": 10000,
   "memory": 4096,
   "timeout": 900,
   "tasks-per-worker": 4
}

for topcoffea, set to number of cores of workers
configuring runtime logs

We recommend to always enable all the logs.

```python
eexecutors_args = {
    'schema': NanoAODSchema,
    'master-name': '{}-wq-coffeal'.format(os.environ['USER']),
    'environment-file': topeftenv.get_environment(),
    'port': 9123,  # or a range [9123, 9130]
    'extra-input-files': ['topeft.py'],

    'cores': 2,  # now resources are maximum allowed
    'memory': 4000,  # in MB
    'disk': 4000,  # in MB

    'resource-monitor': True,
    'resources-mode': 'auto',
    'verbose': True

    'debug-log': 'debug.log',
    'transactions-log': 'tr.log',
    'stats-log': 'stats.log',
}
```
$ grep '\<TASK 1\>' tr.log

1550697985850270 9374 TASK 1 WAITING my-tasks FIRST_RESOURCES {"cores": [1, "cores"]}
1550698004105770 9374 TASK 1 RUNNING 127.0.0.1:40730 FIRST_RESOURCES {"cores": [1, "cores"], "memory": "MB"}
1550698004473367 9374 TASK 1 WAITING RETRIEVAL 127.0.0.1:40730
1550698004475215 9374 TASK 1 RETRIEVED RESOURCE_EXHAUSTION {"disk": [20, "MB"]}
1550698004475384 9374 TASK 1 WAITING my-tasks MAX_RESOURCES {"cores": [1, "cores"], "memory": "MB"}
1550698046053626 9374 TASK 1 RUNNING 127.0.0.1:40734 MAX_RESOURCES {"cores": [1, "cores"], "memory": "MB"}
1550698046444043 9374 TASK 1 WAITING RETRIEVAL 127.0.0.1:40734
1550698046444540 9374 TASK 1 RETRIEVED SUCCESS {"start": [1550698046079981, "us"], "end": [155069804638569, "us"], "memory": [1, 38569, "MB"], "virtual_memory": [6, "MB"], "swap_memory": [0, 0, "MB"], "bytes_written": [0, 0, "MB"], "bytes_received": [0, 0, "MB"], "bytes_sent": [0, 0, "MB"], "bandwidth": [7, 0, "Mbps"], "disk": [20, "MB"], "machine_cpu": [8, "cores"], "machine_load": [0.31, 0.31, "procs"]}
1550698046445762 9374 TASK 1 DONE SUCCESS {"start": [1550698046079981, "us"], "end": [1550698046445762, "us"], "memory": [1, 38569, "MB"], "virtual_memory": [6, "MB"], "swap_memory": [0, 0, "MB"], "bytes_written": [0, 0, "MB"], "bytes_received": [0, 0, "MB"], "bytes_sent": [0, 0, "MB"], "bandwidth": [0, 0, "Mbps"], "disk": [201, "MB"], "machine_cpu": [8, "cores"], "machine_load": [0.31, 0.31, "procs"]}
statistics log

Use `work_queue_graph_log` to visualize the statistics log:

```
$ work_queue_graph_log stats.log
$ display my_stats.*.svn
```
other ways to access statistics

$ work_queue_status -l HOST PORT
{"name":"cclws16.cse.nd.edu","address":"129.74.153.171","tasks_total_disk":0,...
Work Queue API

http://ccl.cse.nd.edu/software/manuals/api/python

http://ccl.cse.nd.edu/software/manuals/api/perl

http://ccl.cse.nd.edu/software/manuals/api/C
thanks!

questions:
btovar@nd.edu

forum:
https://ccl.cse.nd.edu/community/forum

manuals:
http://ccl.cse.nd.edu/software

repositories:
https://github.com/cooperative-computing-lab/cctools
https://github.com/cooperative-computing-lab/makeflow-examples

This work was supported by:

NSF grant ACI 1642609
"SI2-SSE: Scaling up Science on Cyberinfrastructure with the Cooperative Computing Tools"

DOE grant ER26110
"dV/dt - Accelerating the Rate of Progress Towards Extreme Scale Collaborative Science"
extra slides
using the work queue executor: setup

```bash
# install miniconda from #
https://docs.conda.io/en/latest/miniconda.html

$ conda create --yes --name topcoffea-env python=3.8 dill
$ conda activate --yes topcoffea-env
$ conda install --yes -c conda-forge coffea ndcctools xrootd

$ git clone https://github.com/TopEFT/topcoffea.git
$ cd topcoffea
$ pip install -e .

$ cd topcoffea/analysis/topEFT
```
Stand-alone resource monitoring

```
resource_monitor -L"cores: 4" -L"memory: 4096" -- cmd
```

http://ccl.cse.nd.edu/software/manuals/resource_monitor.html

(does not work as well on static executables that fork)