UNIVERSITY OF NOTRE DAME

How Many Eggs Can You Fit in One Nest?

Dynamically Shaping High Throughput Workflows

Douglas Thain, Ben Tovar, and Thanh Son Phung
University of Notre Dame
HTCondor Week, May 23 2022

CCTools

HTCondor at Notre Dame

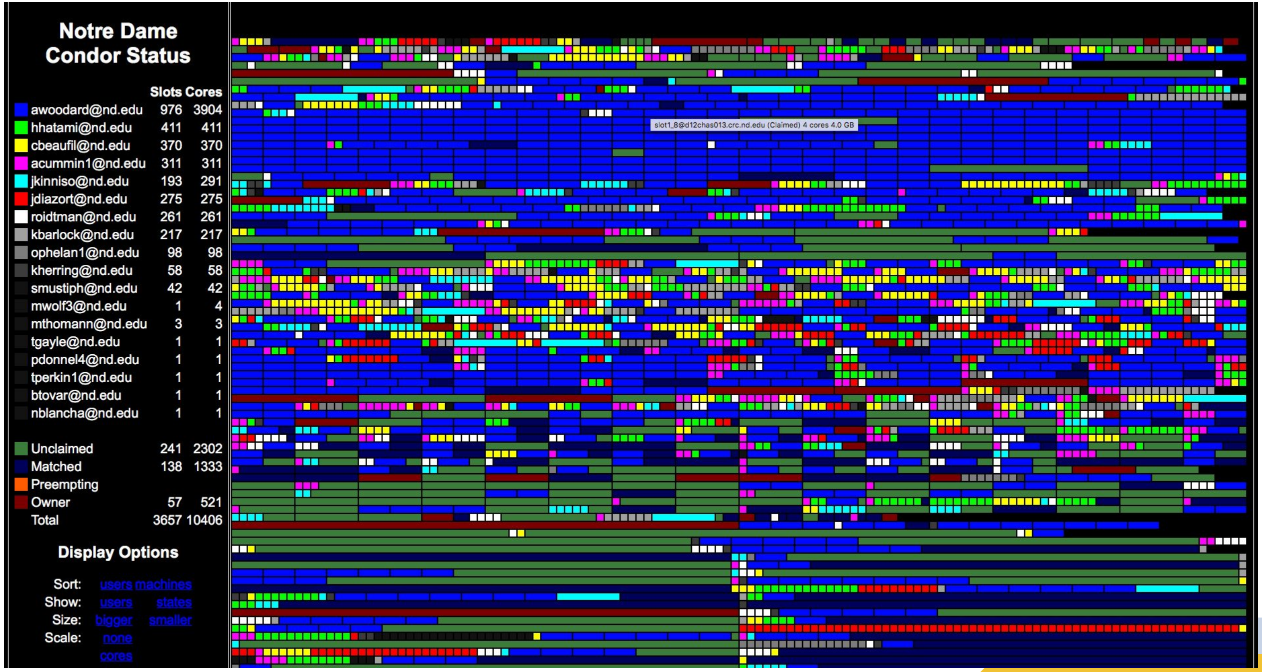
submit

complete

start

complete

Dynamic Workflow Application



https://condor.cse.nd.edu

Dynamic Workflows with Work Queue

CCTools

Work Queue

Work Queue is a framework for building large distributed applications that span thousands of machines drawn from clusters, clouds, and grids. Work Queue applications are written in Python, Perl, or C using a simple API that allows users to define tasks, submit them to the queue, and wait for completion. Tasks are executed by a general worker process that can run on any available machine. Each worker calls home to the manager process, arranges for data transfer, and executes the tasks. A wide variety of scheduling and resource management features are provided to enable the efficient use of large fleets of multicore servers. The system handles a wide variety of failures, allowing for dynamically scalable and robust applications.

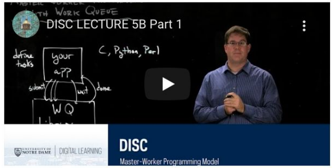
Install Work Queue

Who Uses Work Queue?

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 the Parsl workflow system, the Coffea analysis framework, the the Makeflow workflow engine, SHADHO, Lobster, NanoReactors, ForceBalance, Accelerated Weighted Ensemble, the SAND genome assembler, 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, the University of Wisconsin, and many other locations.

Learn About Work Queue

- Work Queue User's Manual
- Work Queue API (Python | Perl | C)
- Work Queue Example Program (Python | Perl | C)
- Example Application Repository
- Work Queue Status Display
- Getting Help with Work Queue




```
import work_queue as wq

queue = wq.WorkQueue(9123)

for p in range(1,100):
    task = wq.Task("./mysim -p {}".format(p))
    task.specify_memory(1024)
    queue.submit(task)

while not queue.empty():
    task = queue.wait(5)
    if task:
        printf("task {} completed".format(task.id))
```

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

Dynamic Workflows with Work Queue

CCTools

Work Queue

Work Queue is a framework for building large distributed applications that span thousands of machines drawn from clusters, clouds, and grids. Work Queue applications are written in Python, Perl, or C using a simple API that allows users to define tasks, submit them to the queue, and wait for completion. Tasks are executed by a general worker process that can run on any available machine. Each worker calls home to the manager process, arranges for data transfer, and executes the tasks. A wide variety of scheduling and resource management features are provided to enable the efficient use of large fleets of multicore servers. The system handles a wide variety of failures, allowing for dynamically scalable and robust applications.

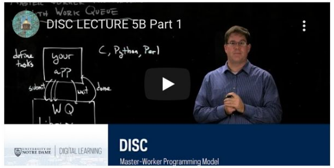
Install Work Queue

Who Uses Work Queue?

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 the Parsl workflow system, the Coffea analysis framework, the the Makeflow workflow engine, SHADHO, Lobster, NanoReactors, ForceBalance, Accelerated Weighted Ensemble, the SAND genome assembler, 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, the University of Wisconsin, and many other locations.

Learn About Work Queue

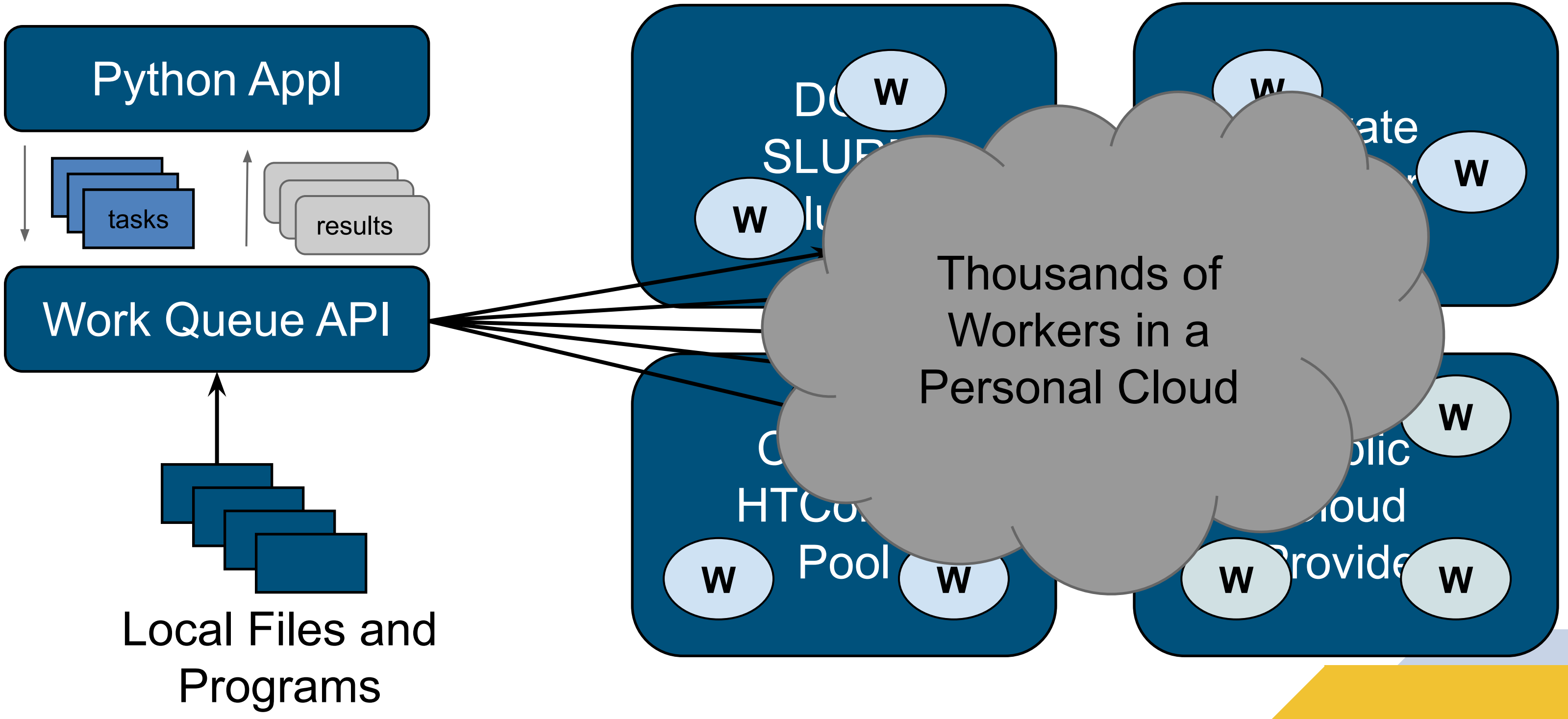
- Work Queue User's Manual
- Work Queue API (Python | Perl | C)
- Work Queue Example Program (Python | Perl | C)
- Example Application Repository
- Work Queue Status Display
- Getting Help with Work Queue




```
import work_queue as wq

queue = wq.WorkQueue(9123)

for p in range(1,100):
    task = wq.PythonTask(myfunc,p)
    task.specify_memory(1024)
    queue.submit(task)

while not queue.empty():
    task = queue.wait(5)
    if task:
        printf("task {} completed".format(task.id))
```

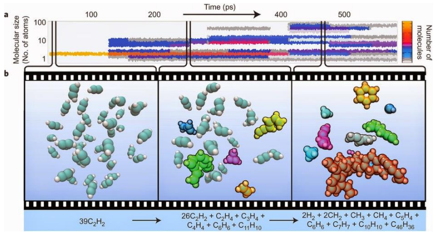
http://ccl.cse.nd.edu/workqueue

Work Queue Architecture

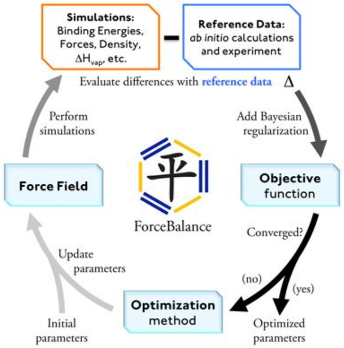


Some Work Queue Applications

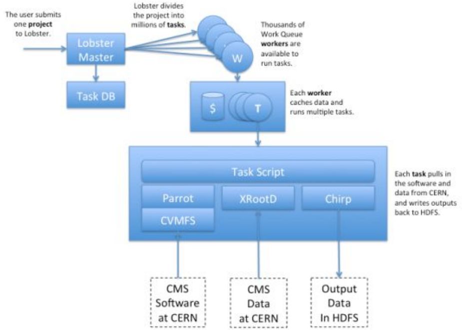
Nanoreactors
ab-initio Chemistry



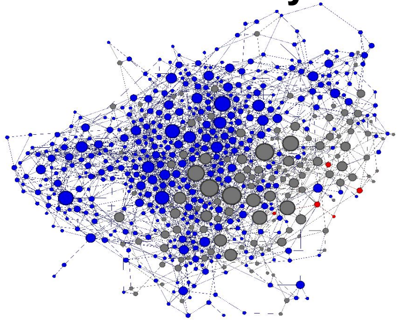
ForceBalance
FF Optimization



Lobster
CMS Data Analysis



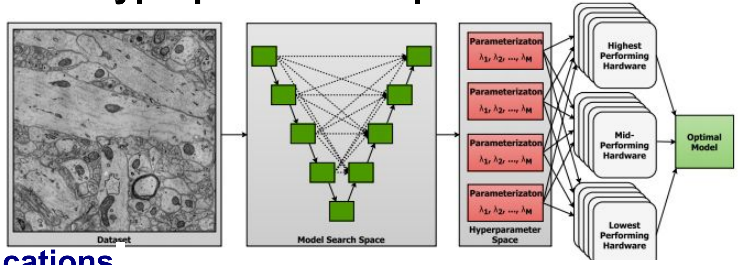
Adaptive Weighted Ensemble
Molecular Dynamics



Low-Level API:

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

SHADHO
Hyperparameter Optimization

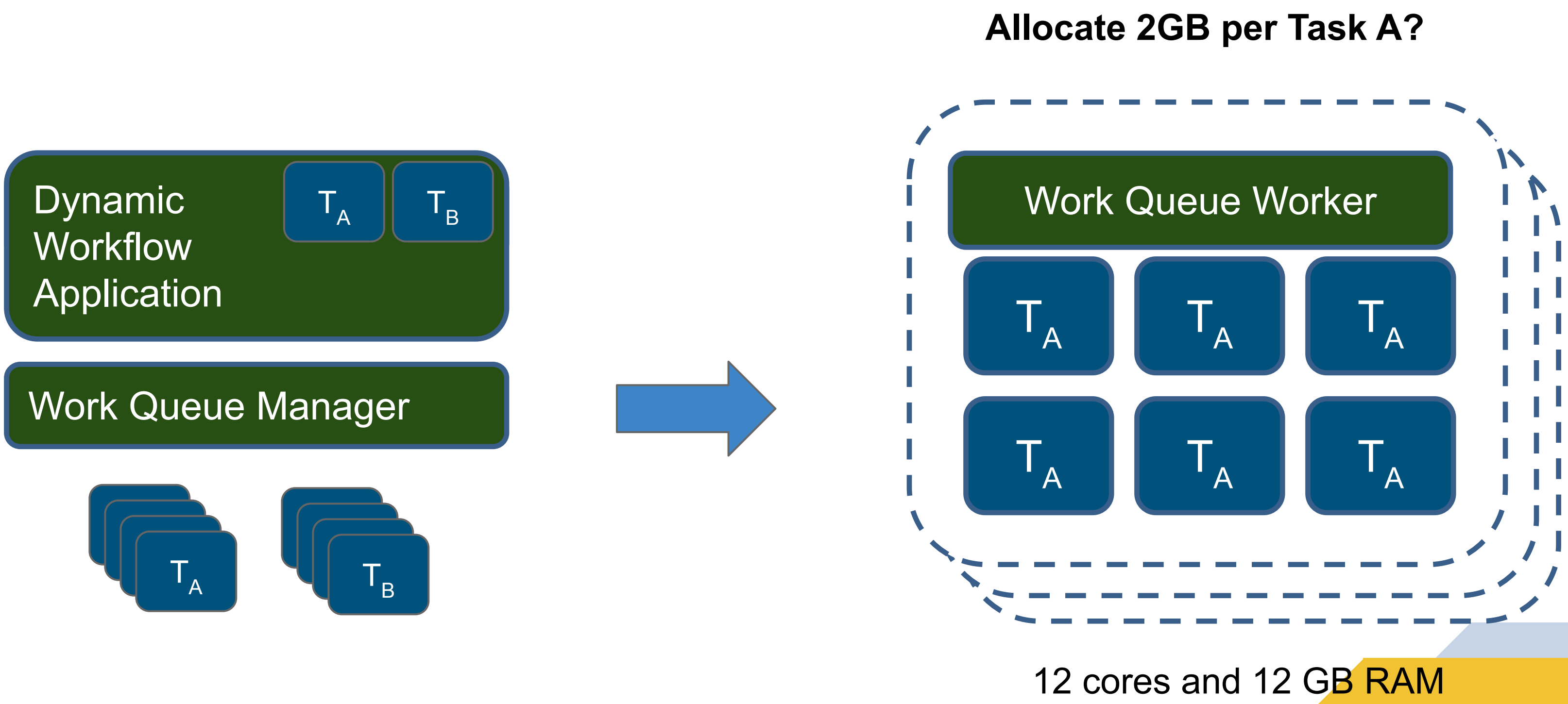


Peter Bui, Dinesh Rajan, Badi Abdul-Wahid, Jesus Izaguirre, Douglas Thain,
Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications,
Workshop on Python for High Performance and Scientific Computing (PyHPC) at Supercomputing 2011.

A simple question:

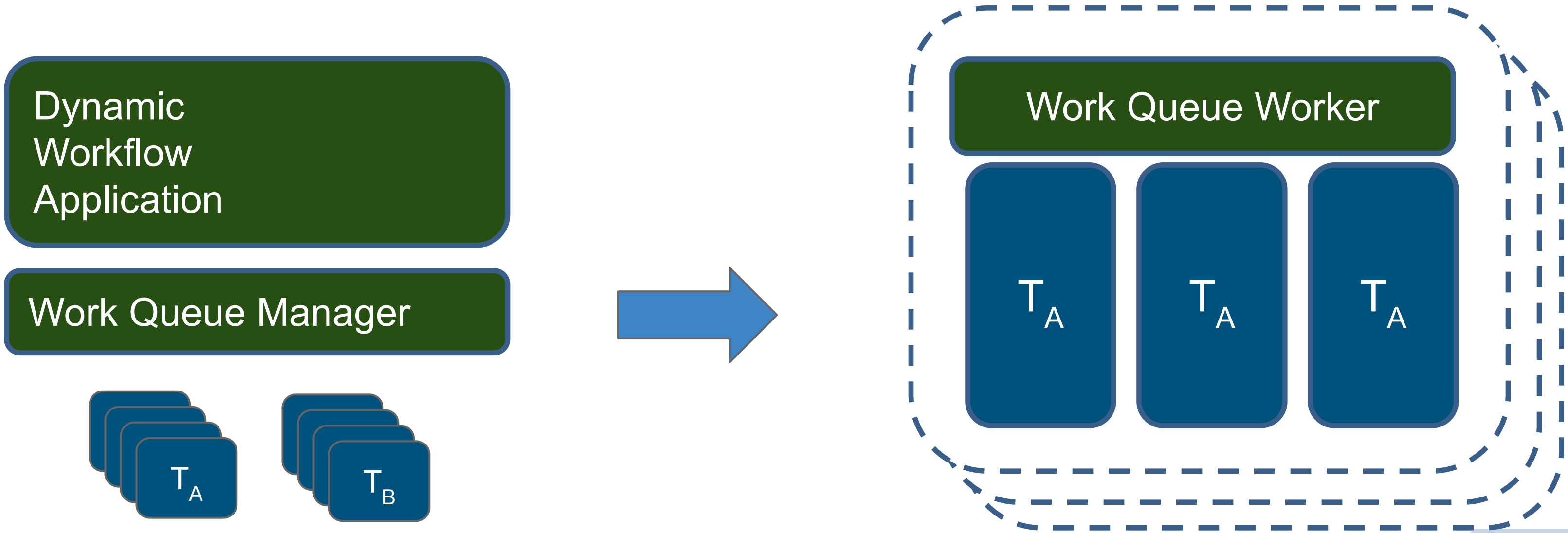
What resources does each task need?
(memory, cores, gpu, . . .)

Packing Tasks Into Manycore Nodes



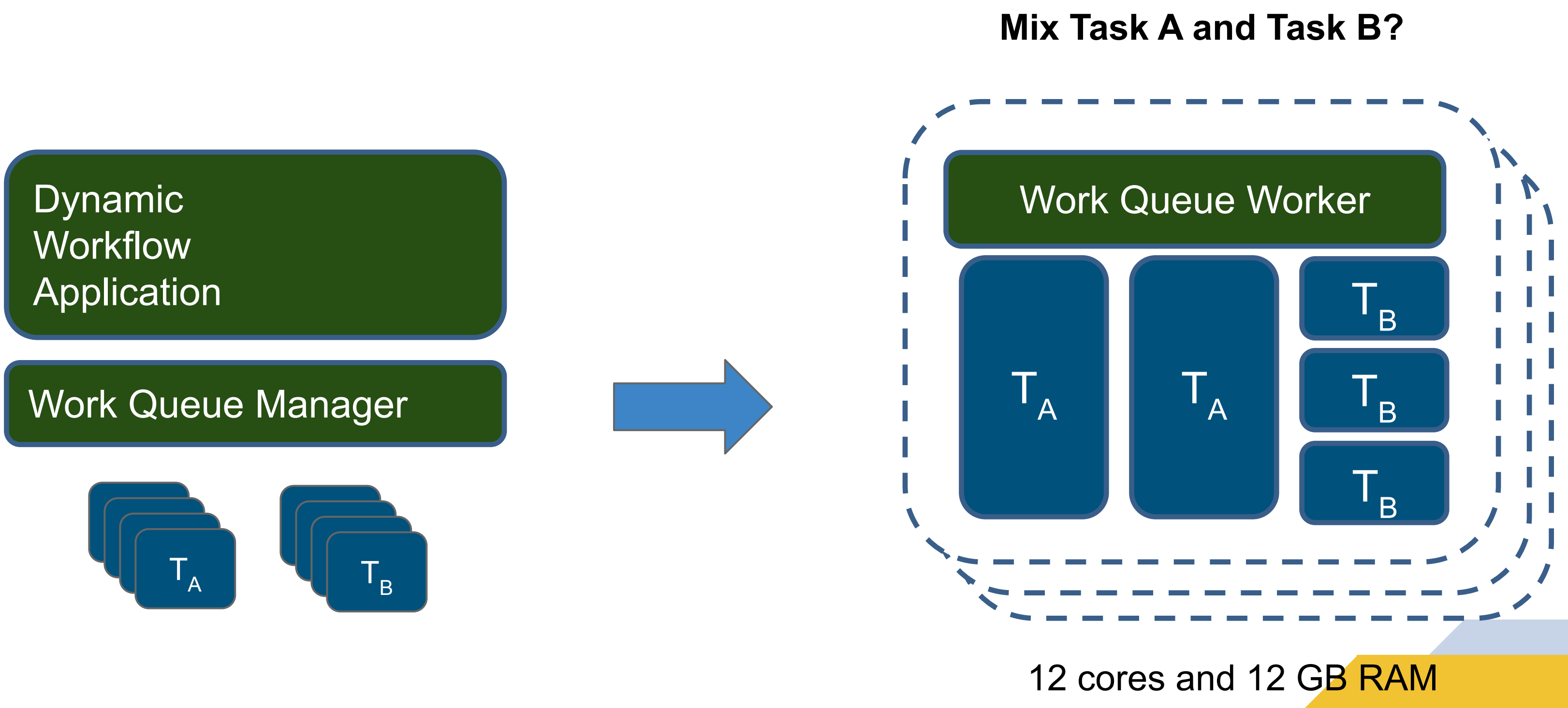
Packing Tasks Into Manycore Nodes

Allocate 4GB per Task A?

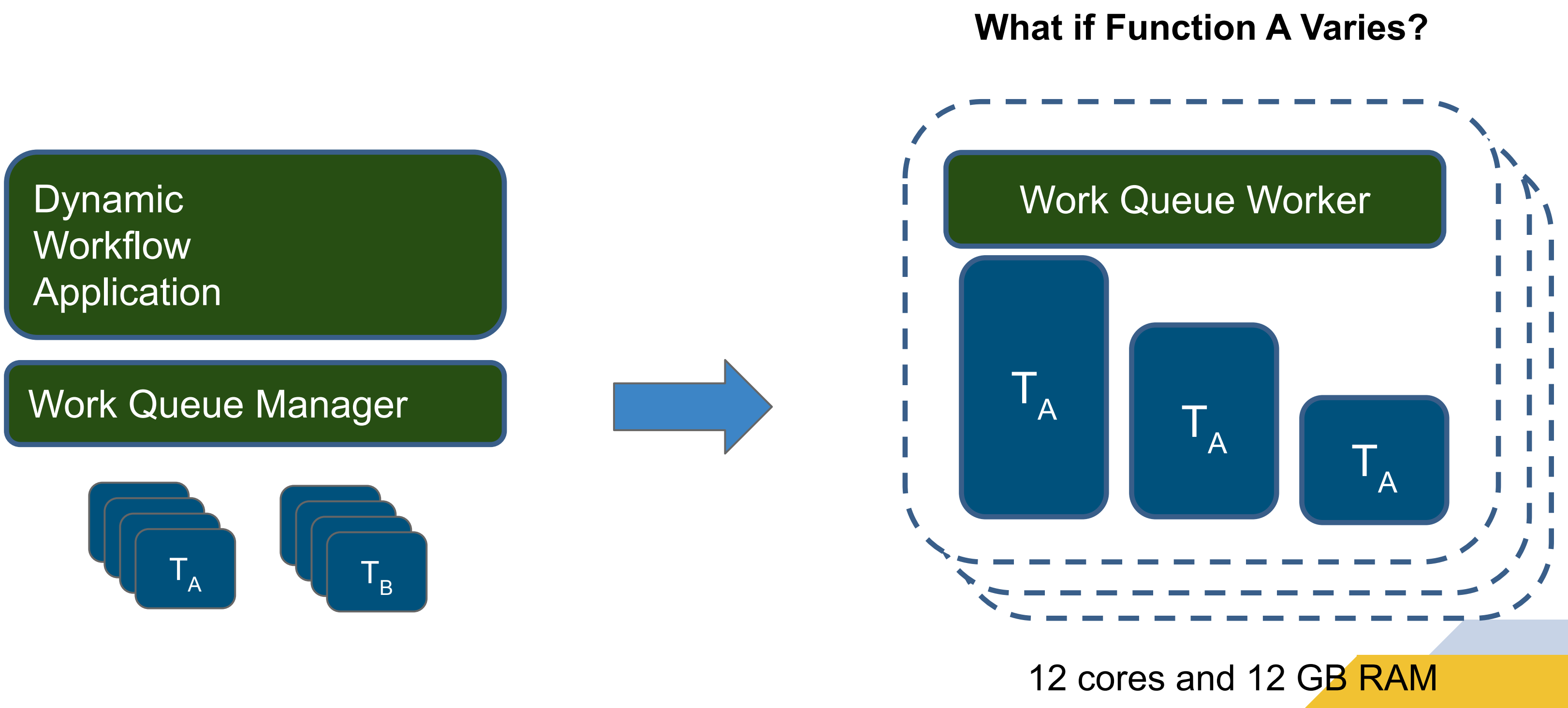


12 cores and 12 GB RAM

Packing Tasks Into Manycore Nodes



Packing Tasks Into Manycore Nodes



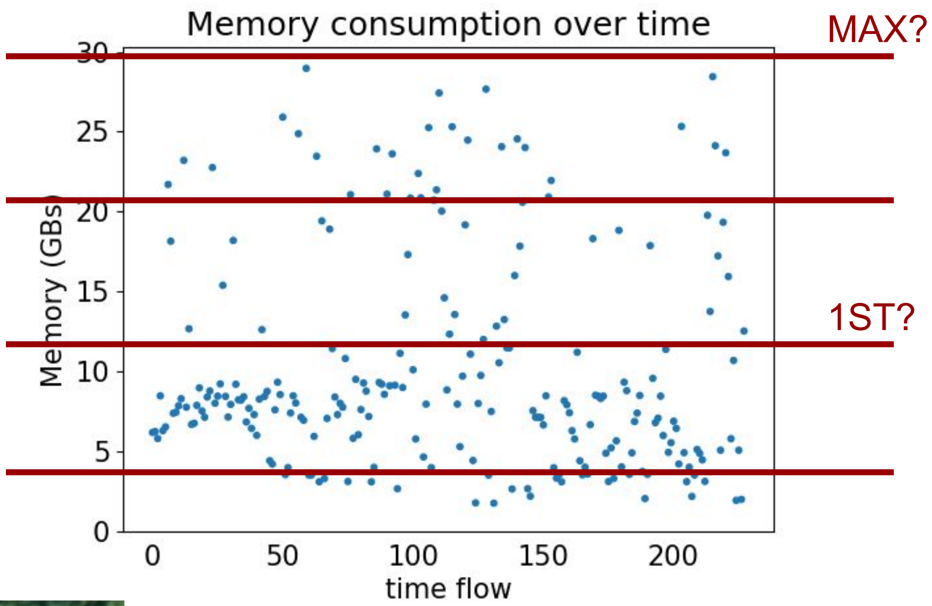
Important Questions for Scheduling Tasks

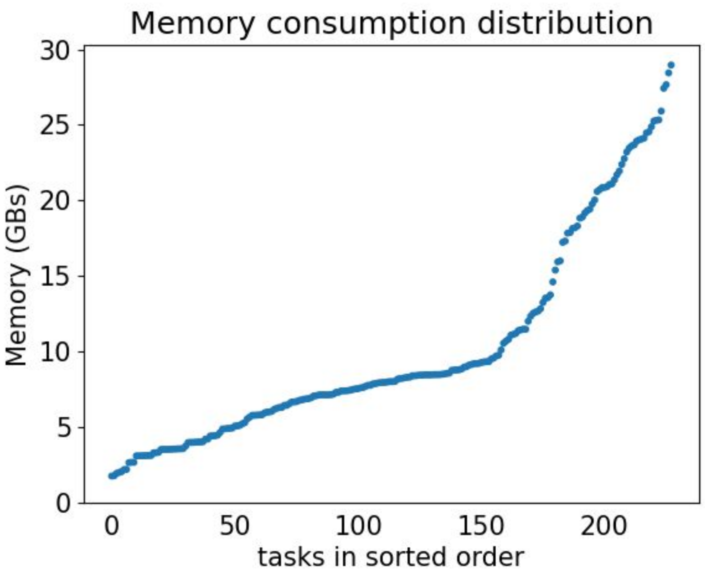
CCTools

We would like the user to tell us:
- How much memory does each task need?
- How many cores does each task need?
- Does this application use any GPUs?

But you might as well ask the user:
- How many roads must a man walk down?

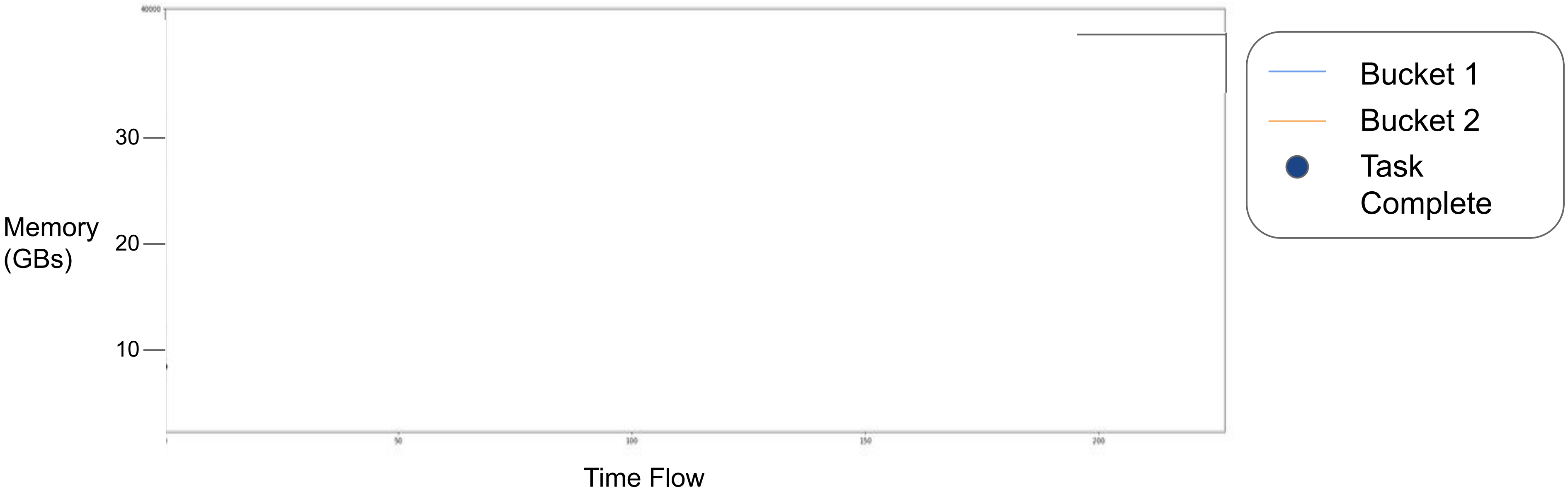
Example: Memory Usage in Colmena-XTB



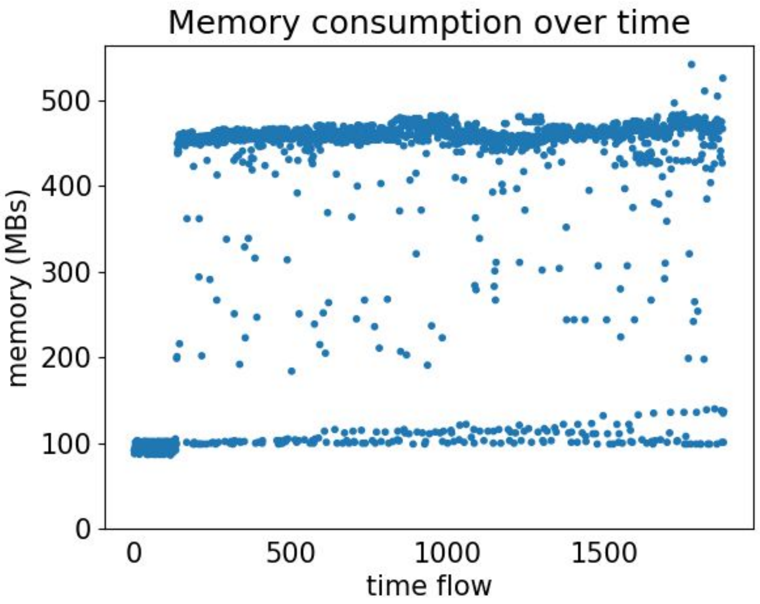


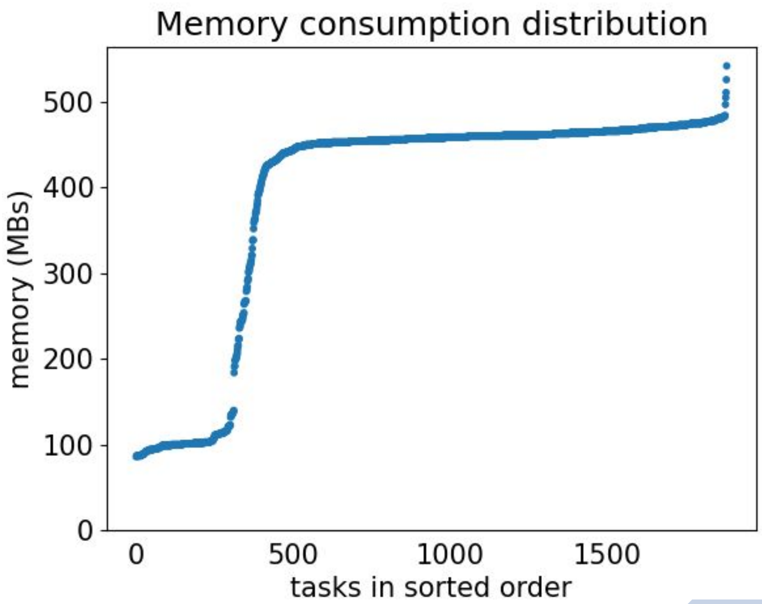
Thanh Son Phung, Logan Ward, Kyle Chard, Douglas Thain, "**Not All Tasks are Created Equal: Adaptive Resource Allocation for Heterogeneous Tasks in Dynamic Workflows"**, WORKS Workshop at Supercomputing 2021.

Dynamic K-means Bucketing

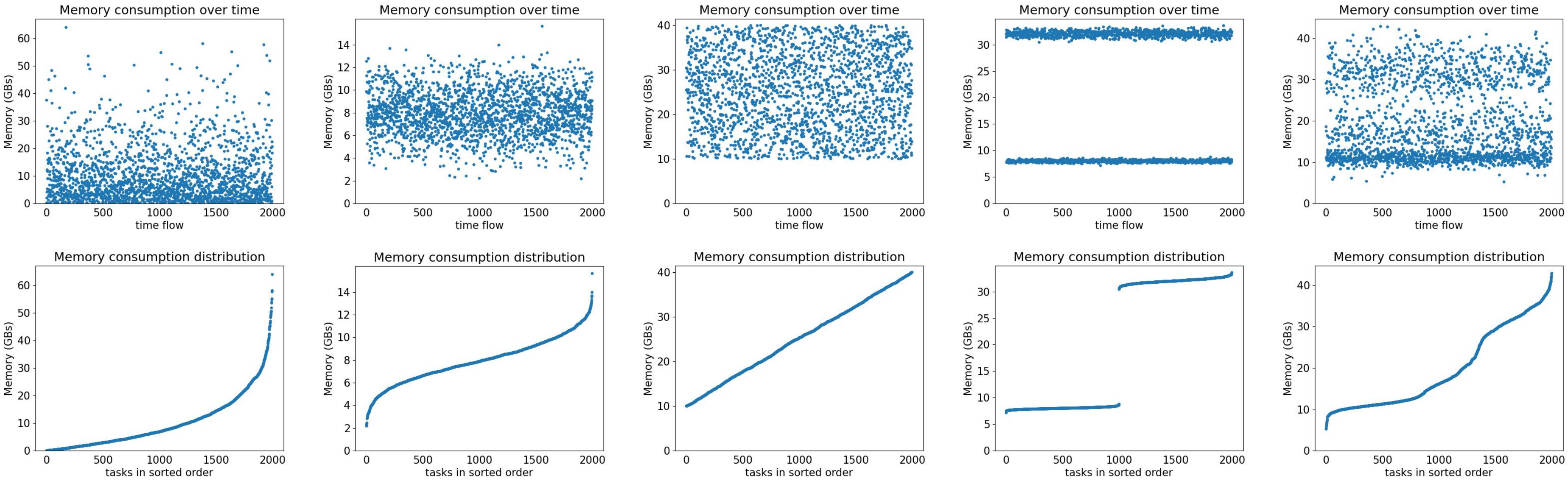


Example: Memory Consumption in TopEFT





Synthetic Distributions



Exponential Normal Uniform Bimodal Trimodal

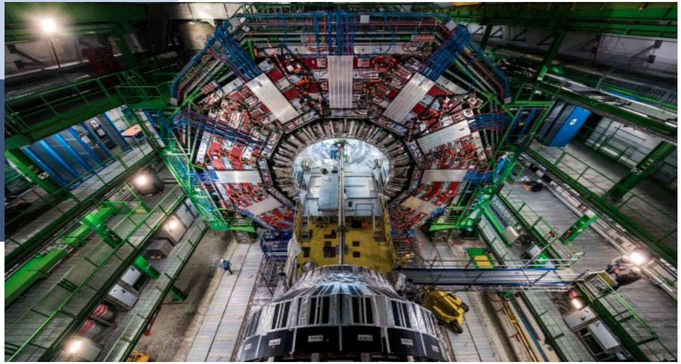
Results

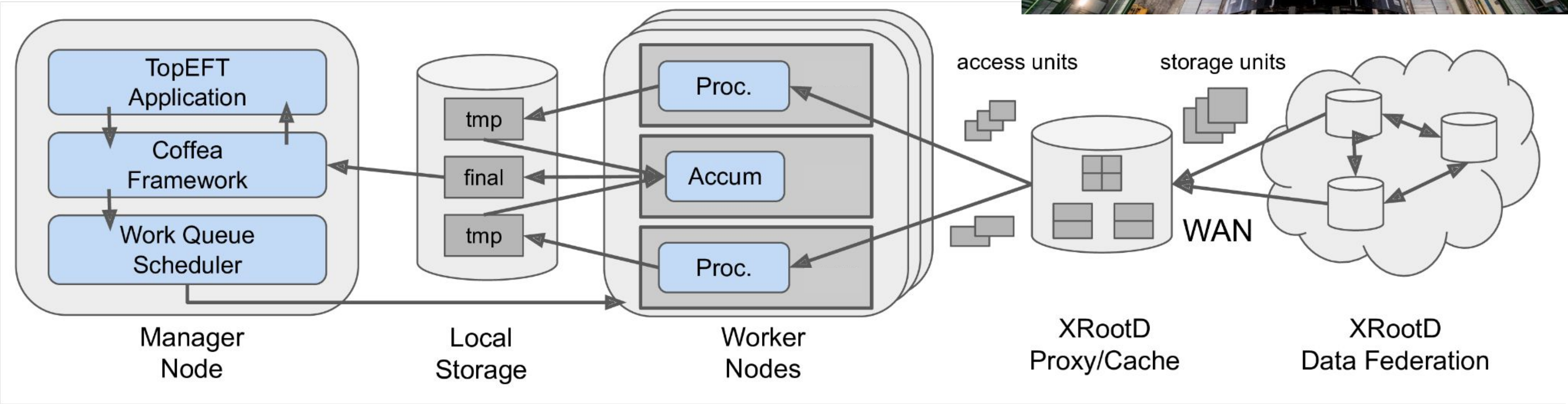
Metric - Average Task Efficiency (ATE): $ATE(S) = \frac{1}{n}\sum_{i=1}^{n}\frac{c_i}{a_i}$

→tracks **the ratio of resources a task uses to resources a task is given**, on average.

	Colmena	TopEFT	Exponential	Uniform	Normal	Bimodal	Trimodal
Whole Machine	15.8	0.6	12.4	39.1	15.7	31.3	30.7
5% above Max	33.2	69.1	48.4	59.6	15.7	56.7	43.7
K-means Bucketing	**43.9***	**91.0**	**56.3**	**62.4**	**16.1**	**93.9**	**71.3**

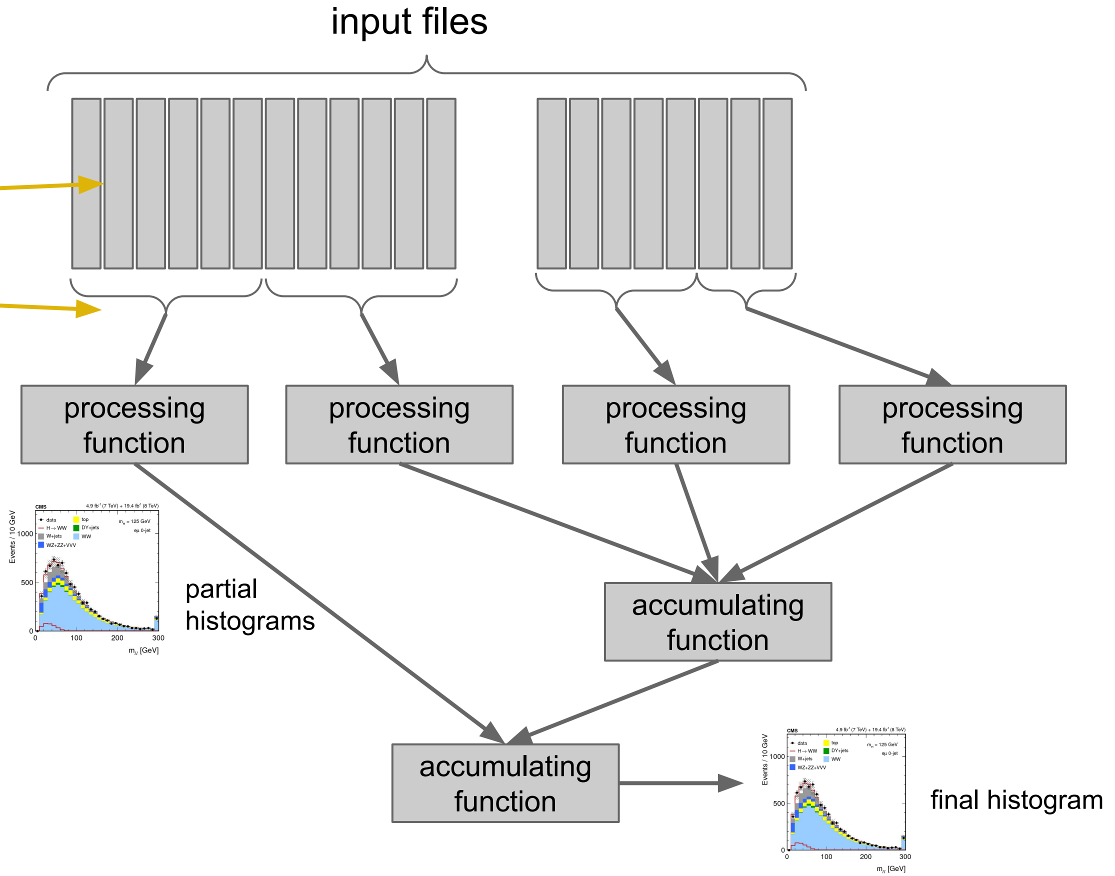
TopEFT CMS Data Analysis Application

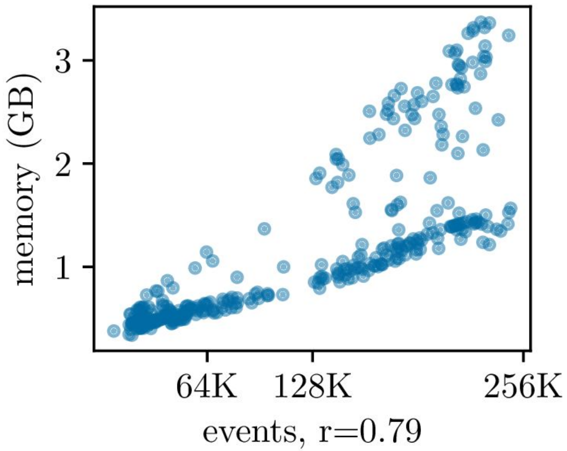




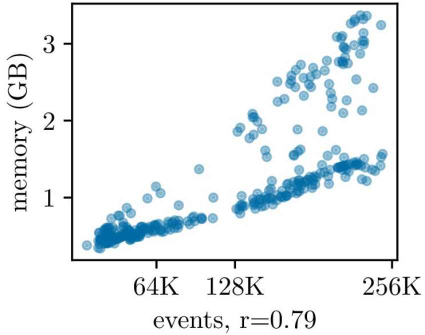
Dynamic Task Shaping for High Throughput Data Analysis Applications in High Energy Physics, Ben Tovar, Ben Lyons, Kelci Mohrman, Barry Sly-Delgado, Kevin Lannon, Douglas Thain, IPDPS 2022.

TopEFT / Coffea Data Splitting Workflow



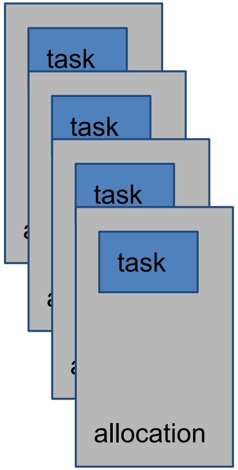


Shaping the Size of Tasks

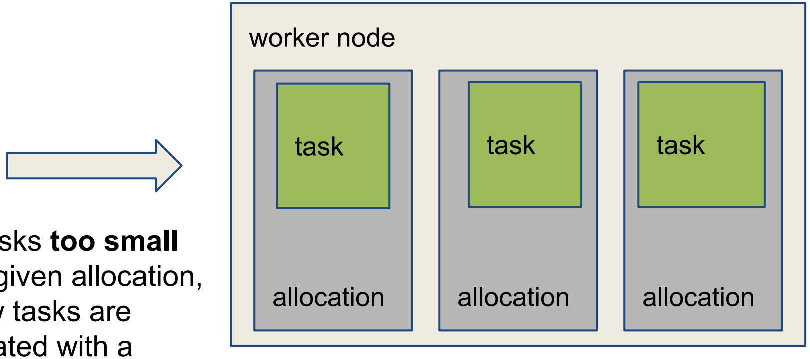


Dynamically update the size of chunks

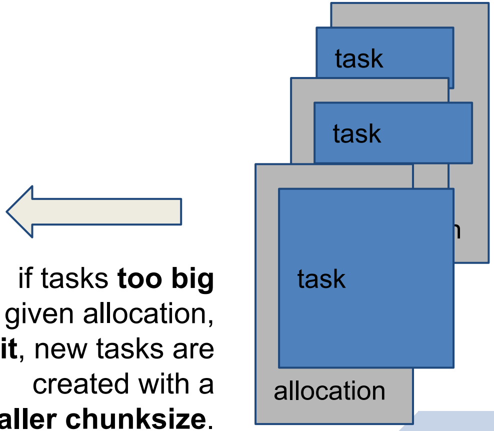
- For a given target resource allocation, modify the size of future chunks according to historical data (e.g., mem vs #events).
- Tasks that exhaust resources are split in two or more.



if tasks **too small** for given allocation, new tasks are created with a **larger chunksize.**

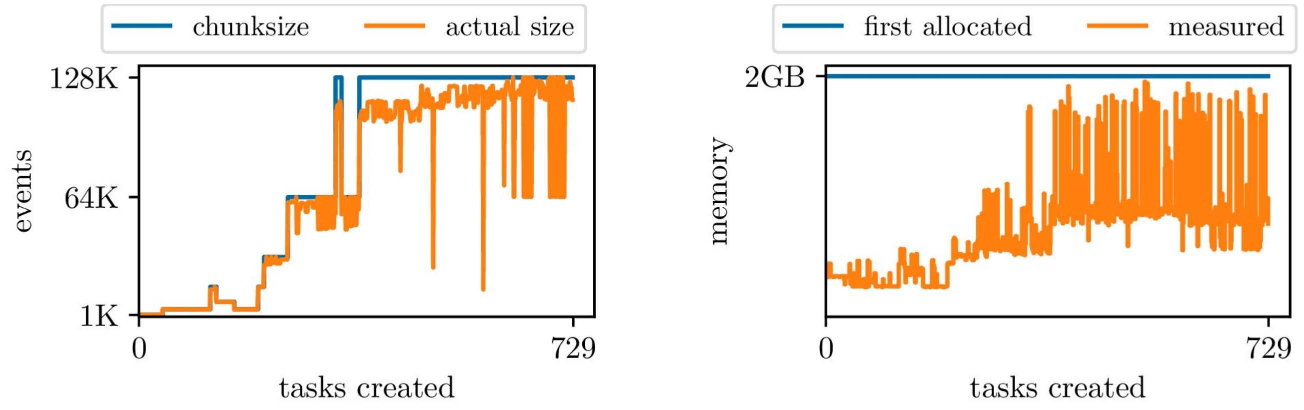


if tasks **too big** for given allocation, **split**, new tasks are created with a **smaller chunksize**.

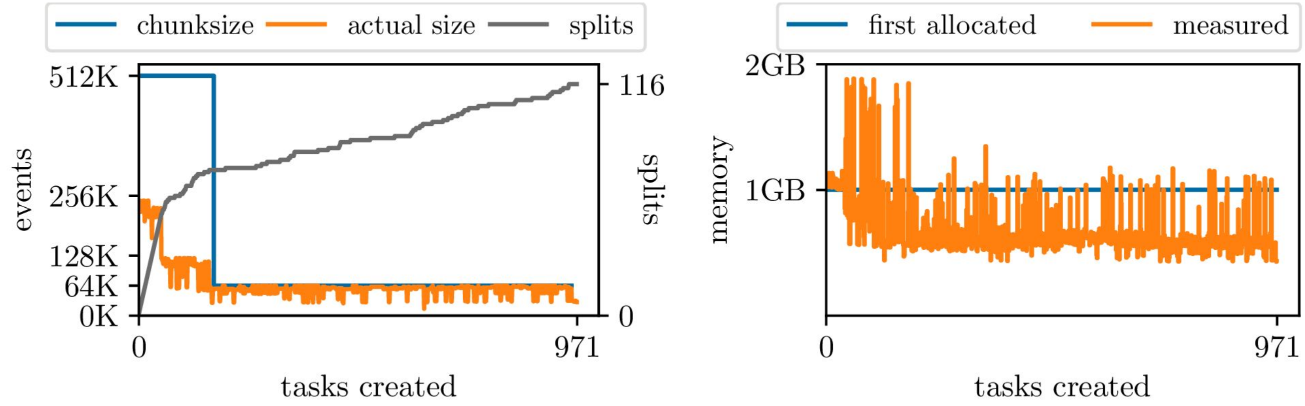


Dynamically Updating Chunksize to Memory Targets

target memory: 2GB
initial chunksize: 1K
user parameters: set A



target memory: 1GB
initial chunksize: 256K
user parameters: set A



How Many Eggs in the Nest?

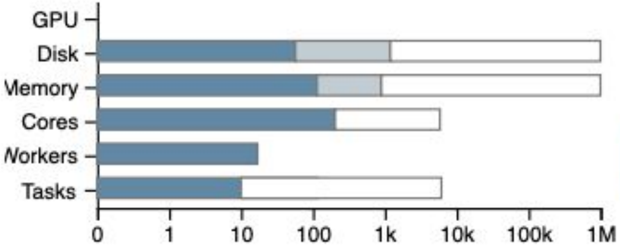
- Applications can have surprisingly complex resource distributions that are unknown to end users.
- **Let the workflow layer aggregate observations and then predict appropriate resource allocations.**
- Fit the nest to the eggs:
  - Dynamically change task **allocations** to fit workers.
- Fit the eggs to the next:
  - Dynamically change task **definitions** to resource targets.
- Continuing challenge: presenting end users with sufficient information to be constructive, but not overwhelming.

How much info to present?

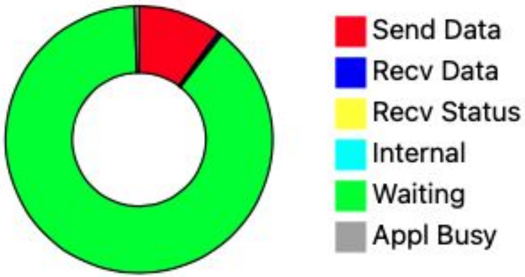
User Interface to Resources?

Host:	earth.crc.nd.edu:9123
Project:	kmohrman-workqueue-coffea
Owner:	kmohrman
Version:	7.4.5 FINAL

Worker Resources



Manager Time



Task Progress

http://ccl.cse.nd.edu/software/workqueue/status

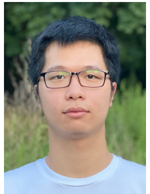
For More Information…

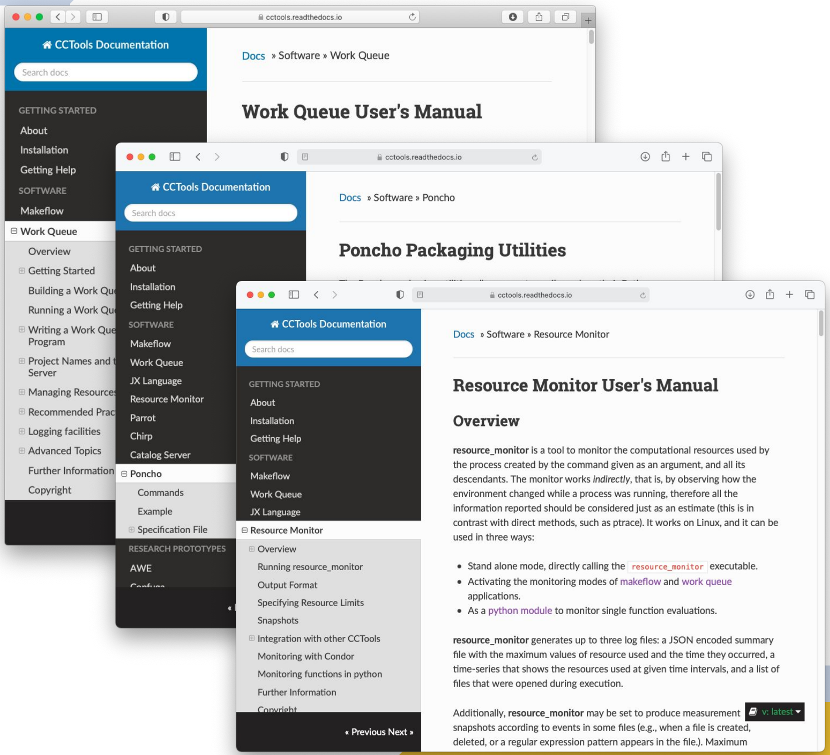
https://cctools.readthedocs.io
https://ccl.cse.nd.edu/software/workqueue

Quick Start:
conda install -c conda-forge ndcctools

dthain@nd.edu btovar@nd.edu tphung@nd.edu



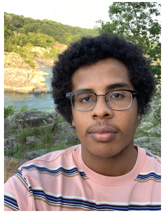




This work was supported by NSF Award OAC-1931348

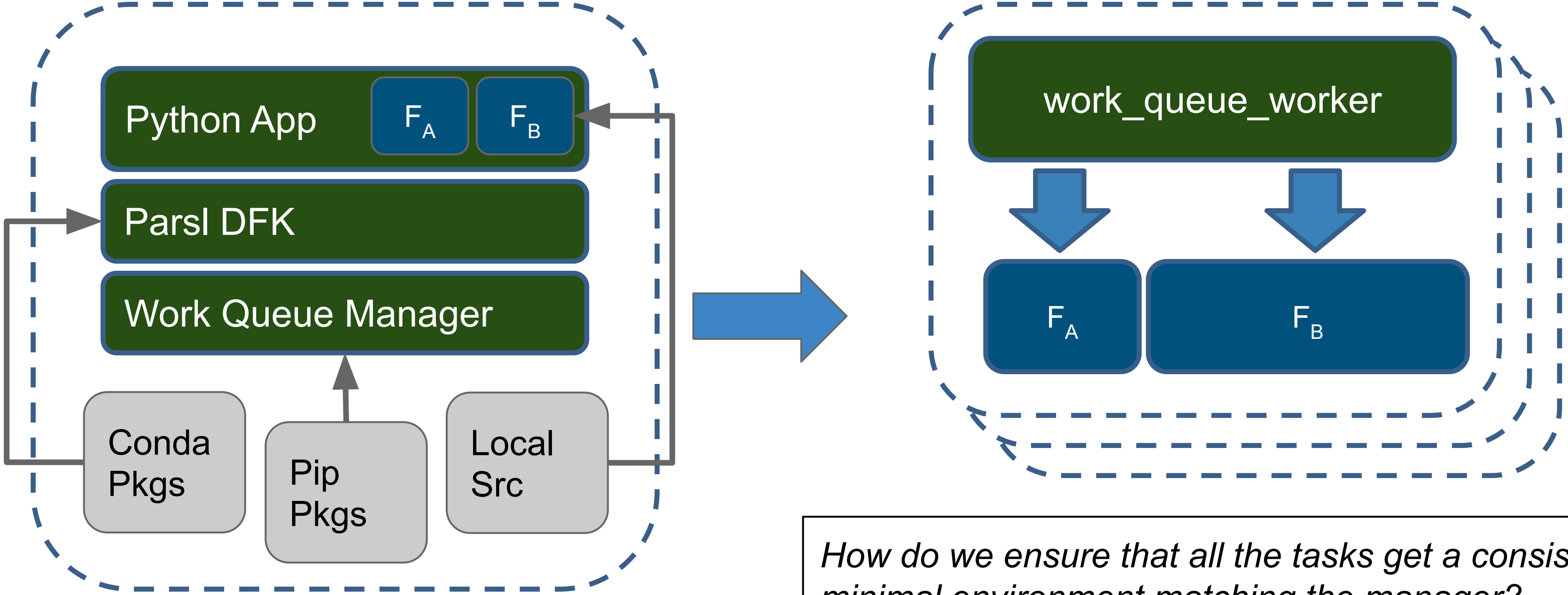
Extra Slides

Run Time Dependency Management



Barry Sly-Delgado

Manager Environment

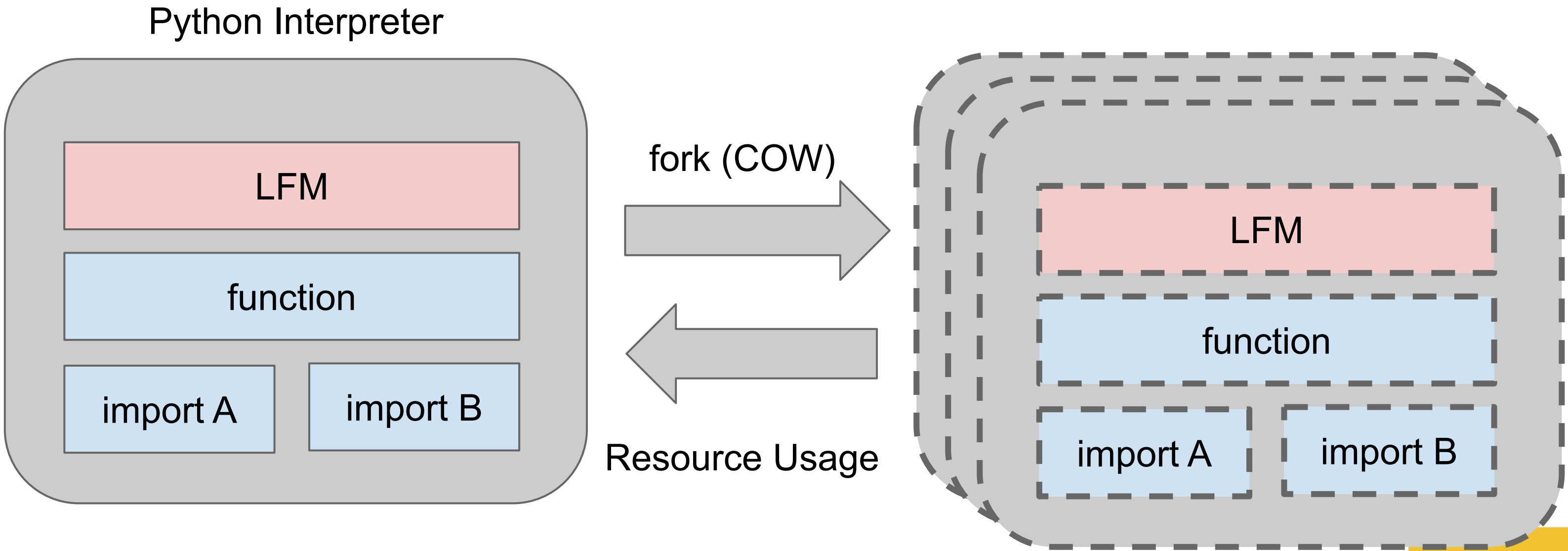


How do we ensure that all the tasks get a consistent, minimal environment matching the manager?

How to measure a single function call?

Ben Tovar

LFM - Lightweight Function Monitor



Tim Shaffer, Zhuozhao Li, Ben Tovar, Yadu Babuji, TJ Dasso, Zoe Surma, Kyle Chard, Ian Foster, and Douglas Thain, **Lightweight Function Monitors for Fine-Grained Management in Large Scale Python Applications**, IEEE International Parallel & Distributed Processing Symposium, May, 2021. DOI: 10.1109/IPDPS49936.2021.00088

Lightweight Function Monitors (LFMs)

Activate LFMs with an import and the @monitored keyword

```
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 'waitied for {} seconds'.format(wait_for)

         (result, resources) = my_function_1(.1)
         print(result, '{}'.format({'memory': resources['memory'], 'wall_time': resources['wall_time']}))

         waitied for 0.1 seconds {'memory': 49, 'wall_time': 101689}
```