Code Profiling and Benchmarking

FITSUM ALEBACHEW

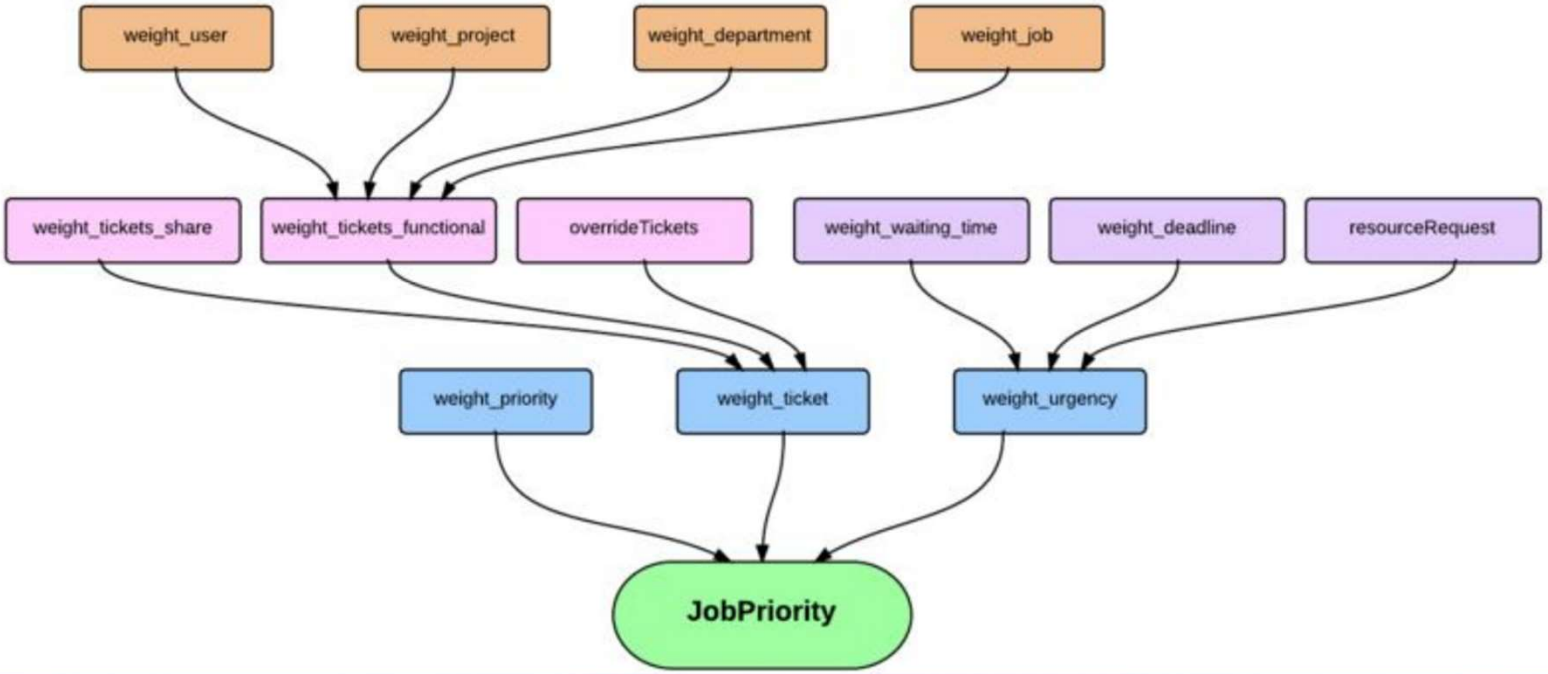
Welcome to the URCF

- Founded in 2014
  - Meet the University's Need for a centralized Research Computing space
  - Shared Condo Computing Model
- URCF Faculty Board – HOLDING ELECTIONS Next Month
  - 3 positions AVAILABLE – please email me or the board to be NOMINATED!!!
  - Chair is Geoff Mainland
- New Rates (Hopefully More Competitive) Coming in July!
- First $100/month/group is FREE! Faculty can apply for GRANTS for Extra Usage! Email Geoff Mainland, me, or ANYONE on the Board

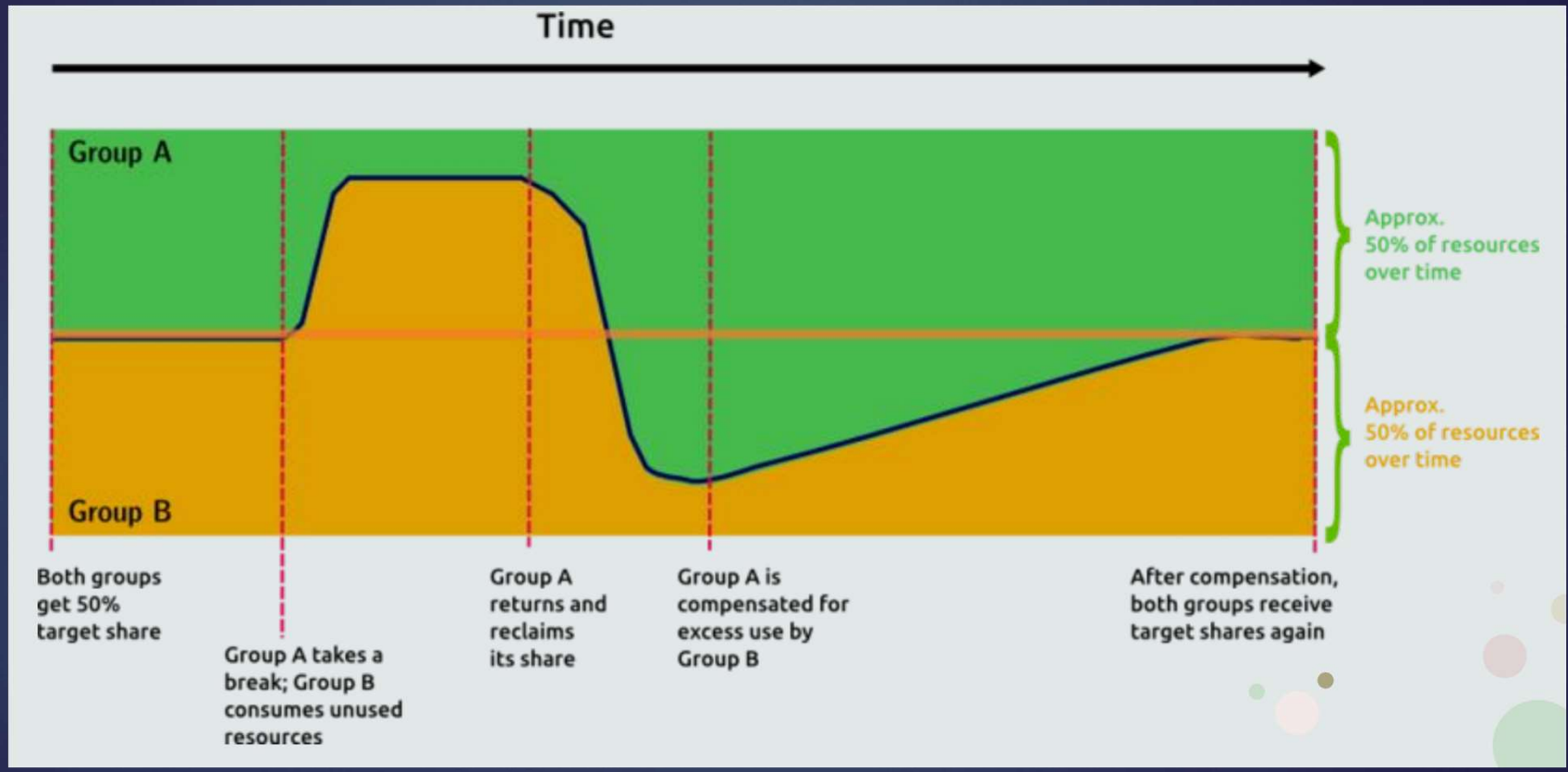
Job Scheduling in SLURM

1. Job Selection - every job in the pending job list is assigned a priority (a scalar value), and the entire list is sorted in order of priority, highest priority first.
2. Job Scheduling - this is where a job is assigned to a set of free resources. The system attempts to find suitable resources for the jobs in priority sequence.

The diagram below shows all the parameters which go into the calculation of a job's priority.



SLURM Resource Allocation



Profiling vs. Benchmarking

Benchmarking and profiling are often used interchangeably, but they are not the same:

- Benchmarking is measuring the overall runtime of a program
  - Usually a single number result
  - Can be used to test a program's efficiency with different parameters
  - Also used for testing speeds of different hardware
- In shared HPC clusters, you can benchmark your code to request the right amount of resources and not waste allocations.

Usage Rates

Picotte Usage Rates

Compute

Compute resource rate: **$0.0123 per SU**

Resources:

- standard compute nodes have 48 cores per node; there are 74 nodes in total
- big memory nodes have 1.5 TiB of memory (RAM) per node; there are 2 nodes in total
- GPU nodes have 4 GPU devices (cards) per node; there are 12 nodes in total

Picotte Compute Rates		
Resource type	**Slurm partition**	**SU per unit resource**
Std. compute	def	1 per core-hour
Big memory	bm	68 per TiB-hour
GPU	gpu	43 per GPU device-hour

Example: Using all 4 GPU devices on a GPU node for 1 hour consumes 172 SU, for a total charge of $0.0123 * 172 = $2.12

NOTE: all resource usage above is computed based on resources reserved for the actual lifetime of a job. E.g. a job requests 4 GPU devices for 1 hour, but runs only on one GPU device for 1 hour. While the actual usage is 1 GPU-hour, the resources set aside are 4 GPU-hours. The billable amount is 4 GPU-hours = 172 SU. This is because those resources are made unavailable to others.

Persistent Storage

Storage rate: ~~**1.48 SU per TiB-hour**~~ **1081 SU per TiB-month**

To compare to Proteus (see above), this is equivalent to ~~$3.06 per TiB-week~~ $13.30 per TiB-month ~= $3.32 per TiB-week.

Example: storing 5 TiB of data for 1 month → $0.0123 * 1081 * 5 = $66.48

Benchmarking

So how do we benchmark our code on Picotte?

Benchmarking

Just let SLURM do it for you! SLURM automatically collects and saves many metrics related to every job it runs. There are several commands you can use:

- seff <jobid>: get efficiency statistics of a job

```
[picotte001] ~$ seff 2588667
Job ID: 2588667
Cluster: picotte
User/Group: fa496/vtune
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:56
CPU Efficiency: 6.69% of 00:13:57 core-walltime
Job Wall-clock time: 00:13:57
Memory Utilized: 4.39 GB
Memory Efficiency: 43.95% of 10.00 GB
[picotte001] ~$
```

Benchmarking (cont.)

- sacct: show details about jobs ran by a user (can use -j <jobid> option)
  - Can display all your recent jobs together
  - Can be formatted with the --format (-o) option

```
[picotte001] CT_Multi_Genus_Data$ sacct -j 2588667
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
2588667      CT_Multi_+        gpu rosenmrip+          1  COMPLETED      0:0
2588667.bat+      batch            rosenmrip+          1  COMPLETED      0:0
2588667.ext+     extern            rosenmrip+          1  COMPLETED      0:0
[picotte001] CT_Multi_Genus_Data$ sacct -o "JobID%17,JobName%15,Partition%4,NodeList%6,Elapsed,State,ExitCode%4,ReqMem%5,MaxRSS,MaxVMSize,AllocTRES%32,AllocGRES%8" -j 2588667
           JobID         JobName Part NodeLi    Elapsed      State Exit ReqMe     MaxRSS  MaxVMSize                          AllocTRES AllocGRE
----------------- --------------- ---- ------ ---------- ---------- ---- ----- ---------- ---------- -------------------------------- --------
          2588667 CT_Multi_Genus+  gpu gpu001   00:13:57  COMPLETED  0:0  10Gn                       billing=172,cpu=1,gres/gpu=4,no+    gpu:4
    2588667.batch           batch      gpu001   00:13:57  COMPLETED  0:0  10Gn   4608040K  61533548K               cpu=1,mem=0,node=1    gpu:4
   2588667.extern          extern      gpu001   00:13:57  COMPLETED  0:0  10Gn       700K    217044K billing=172,cpu=1,gres/gpu=4,no+    gpu:4
[picotte001] CT_Multi_Genus_Data$
```

  - MaxRSS (Resident Set Size) variable above is the total RAM used by your job

```
Fields available:

Account             AdminComment        AllocCPUS           AllocGRES
AllocNodes          AllocTRES           AssocID             AveCPU
AveCPUFreq          AveDiskRead         AveDiskWrite        AvePages
AveRSS              AveVMSize           BlockID             Cluster
Comment             Constraints         ConsumedEnergy      ConsumedEnergyRaw
CPUTime             CPUTimeRAW          DBIndex             DerivedExitCode
Elapsed             ElapsedRaw          Eligible            End
ExitCode            Flags               GID                 Group
JobID               JobIDRaw            JobName             Layout
MaxDiskRead         MaxDiskReadNode     MaxDiskReadTask     MaxDiskWrite
MaxDiskWriteNode    MaxDiskWriteTask    MaxPages            MaxPagesNode
MaxPagesTask        MaxRSS              MaxRSSNode          MaxRSSTask
MaxVMSize           MaxVMSizeNode       MaxVMSizeTask       McsLabel
MinCPU              MinCPUNode          MinCPUTask          NCPUS
NNodes              NodeList            NTasks              Priority
Partition           QOS                 QOSRAW              Reason
ReqCPUFreq          ReqCPUFreqMin       ReqCPUFreqMax       ReqCPUFreqGov
ReqCPUS             ReqGRES             ReqMem              ReqNodes
ReqTRES             Reservation         ReservationId       Reserved
ResvCPU             ResvCPURAW          Start               State
Submit              Suspended           SystemCPU           SystemComment
Timelimit           TimelimitRaw        TotalCPU            TRESUsageInAve
TRESUsageInMax      TRESUsageInMaxNode  TRESUsageInMaxTask  TRESUsageInMin
TRESUsageInMinNode  TRESUsageInMinTask  TRESUsageInTot      TRESUsageOutAve
TRESUsageOutMax     TRESUsageOutMaxNode TRESUsageOutMaxTask TRESUsageOutMin
TRESUsageOutMinNode TRESUsageOutMinTask TRESUsageOutTot     UID
User                UserCPU             WCKey               WCKeyID
WorkDir
```

Benchmarking (cont.)

For runtime of simple commands or scripts:

- time [command]
  - user: time for user code to run (no system calls or tasks)
  - sys: time for system calls and tasks (e.g. memory allocation)

```
[picotte001] demos$ time python time_test.py

real    0m2.087s
user    0m2.055s
sys     0m0.015s
```

What is Profiling

Profiling is measuring the runtime costs of individual components of a program

- Multiple results for individual parts of the code being profiled
- Aimed at finding bottlenecks in code
- Can be used to make your code run more efficiently

Similar concepts with benchmarking, but different applications!

Profilers

Different programming languages have different profilers:

- C/C++ have gprof, valgrind
- Python has cProfile, memory_profiler, line_profiler
- R has lineprof

Python cProfile

cProfile is a very well-known built-in profiler for python.

- It measures the runtime of every function/system call within the code
- python –m cProfile [-o outputfile] [-s sort_order] (-m module| script.py)

```
Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  26/5    0.000    0.000    0.051    0.010 <frozen importlib._bootstrap>:1002(_find_and_load)
     3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:1033(_handle_fromlist)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:112(release)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:152(__init__)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:156(__enter__)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:160(__exit__)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:166(_get_module_lock)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:185(cb)
  36/6    0.000    0.000    0.043    0.007 <frozen importlib._bootstrap>:220(_call_with_frames_removed)
   428    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:231(_verbose_message)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:241(_requires_builtin_wrapper)
    15    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:35(_new_module)
    26    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:351(__init__)
    39    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:385(cached)
    25    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:398(parent)
```

cProfile

- ncalls: number of times function was called
- tottime: total time taken by function (sub-calls excluded)
- percall: tottime/ncalls (rounded down)
- cumtime: tottime + sub-calls (total time taken to complete)
- 2nd percall: cumtime/primitive cells

More details: https://docs.python.org/3/library/profile.html

Some Advice

- Loops are slow
  - Avoid with built-in functions if possible
- Use faster libraries if possible
  - Some libraries offer superior speeds for certain uses
  - numpy has the ability to use multiple cores

Questions? Thank You for Coming!

- Feel free to attend my office hours every weekday 2 - 3 pm (any changes will be reflected on the URCF wiki main page): https://proteusmaster.urcf.drexel.edu/urcfwiki/index.php/Main_Page#Talks_and_Workshops