Use the UMassCS Swarm2 cluster efficiently for your research!

Keen Sung

February 5, 2019

Objectives

- Learn the basic architecture of swarm
- Walk through how to parallelize and run a job
- Tips for optimizing
- Checkpointing
- Troubleshooting

Swarm v Gypsum

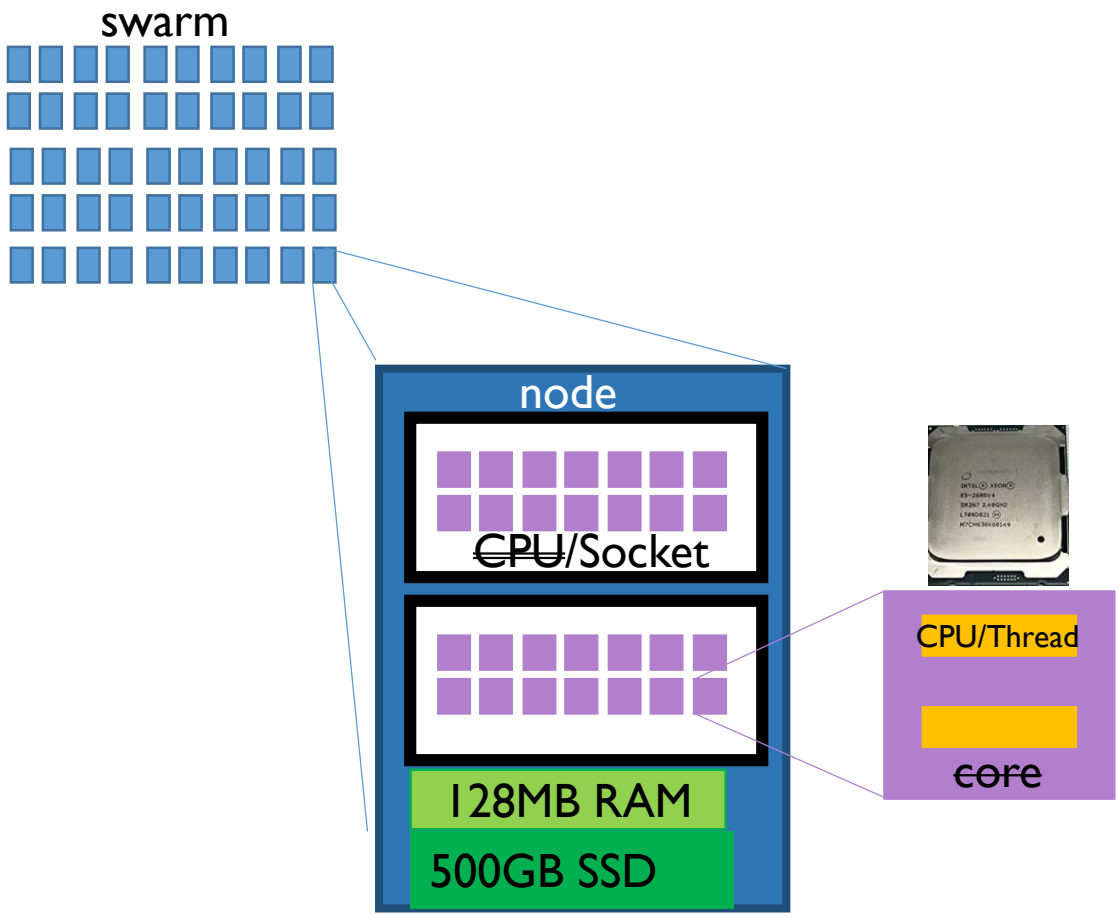
Swarm

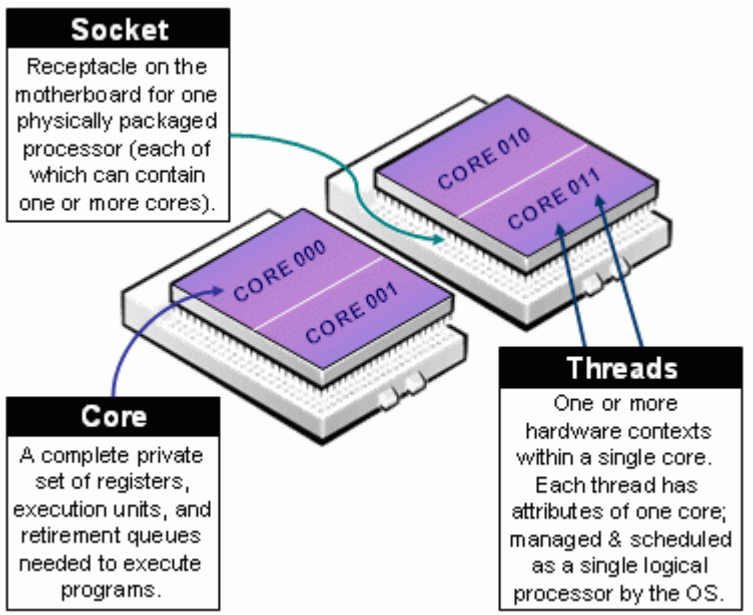
- 50 nodes
  - 56 cores
  - 128 GB RAM
- Total
  - 2800 cores
  - 6.4TB RAM

Gypsum

- 100 nodes
  - 4 GPU (25: M40, 75: TITAN X)
  - 24 cores
  - 256 GB RAM
- 53 nodes
  - 8 GPU (1080Ti)
  - 48 cores
  - 384 GB RAM
- Total
  - 4944 cores
  - 30.72TB RAM
  - 824 GPU

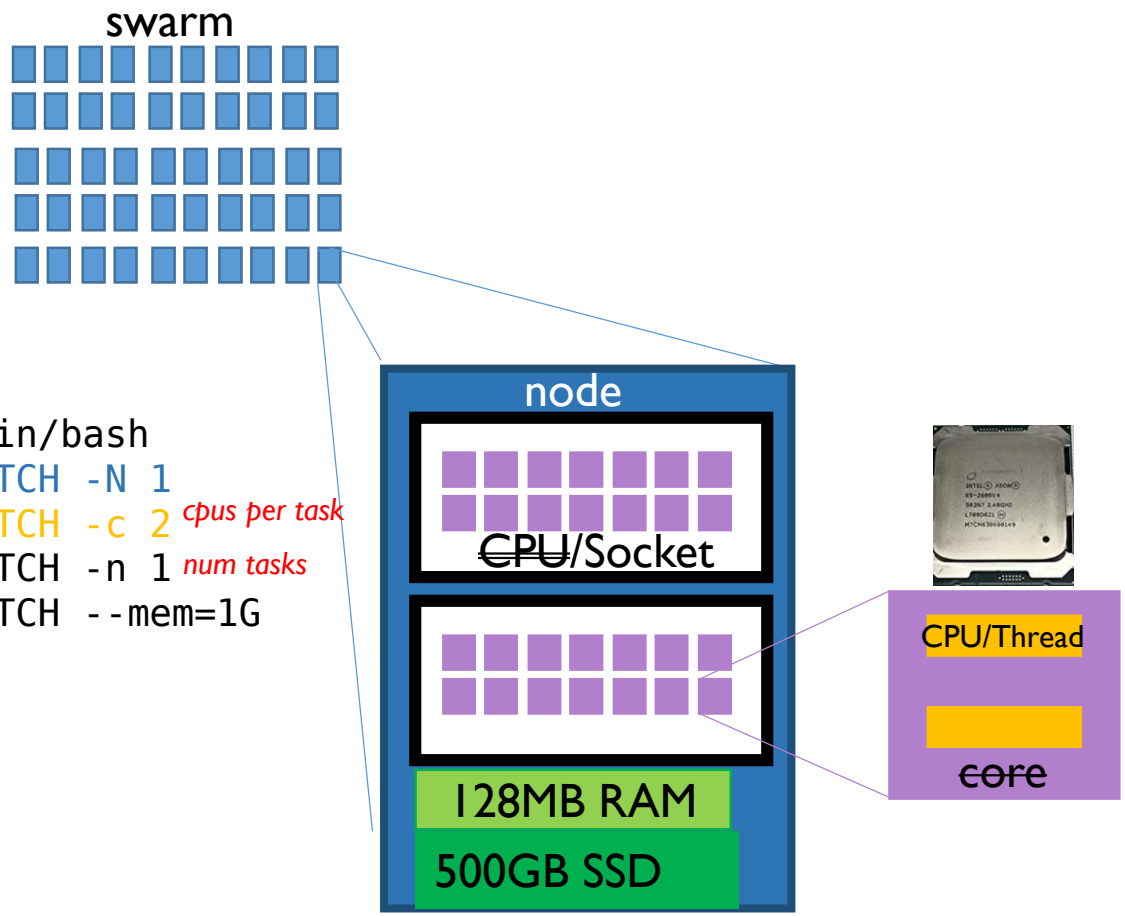
Clarifying ambiguous terminology



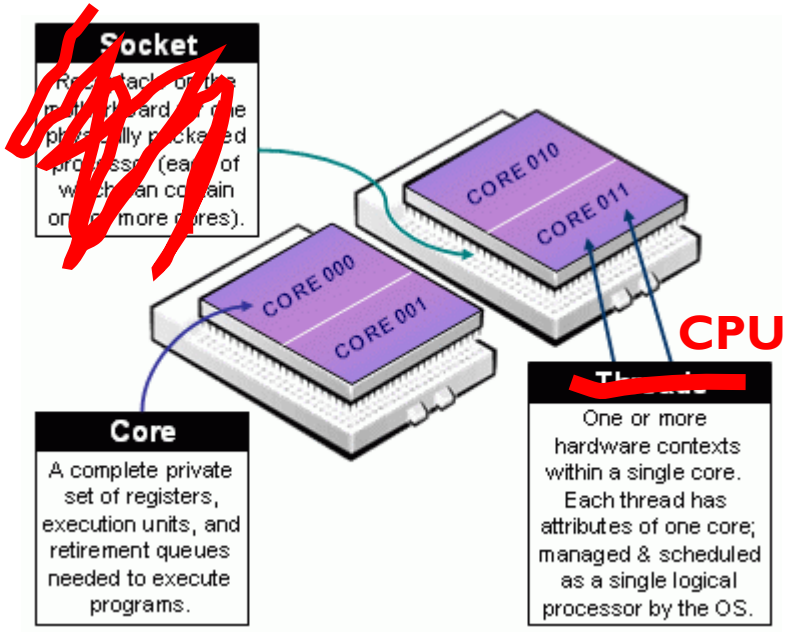


https://slurm.schedmd.com/mc_support.html

Clarifying ambiguous terminology




```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 2    cpus per task
#SBATCH -n 1    num tasks
#SBATCH --mem=1G
```



https://slurm.schedmd.com/mc_support.html

SLURM

- Queuing and scheduling system
- Tries to account for fairness
  - Priority queue based on a fairness score calculated by current and historical usage of **CPU** or **RAM** by you and your group, and the **age** of submission

Table 1. Backfill algorithm pseudocode.

1. Find the shadow time and extra nodes
   1. Sort the list of running jobs according to their expected termination time
   2. Loop over the list and collect nodes until the number of available nodes is sufficient for the first job in the queue
   3. The time at which this happens is the *shadow time*
   4. If at this time more nodes are available than needed by the first queued job, the ones left over are the extra nodes
2. Find a backfill job
   1. Loop on the list of queued jobs in order of arrival
   2. For each one, check whether either of the following conditions hold:
      - It requires no more than the currently free nodes, and will terminate by the shadow time, or
      - It requires no more than the minimum of the currently free nodes and the extra nodes
   3. The first such job can be used for backfilling

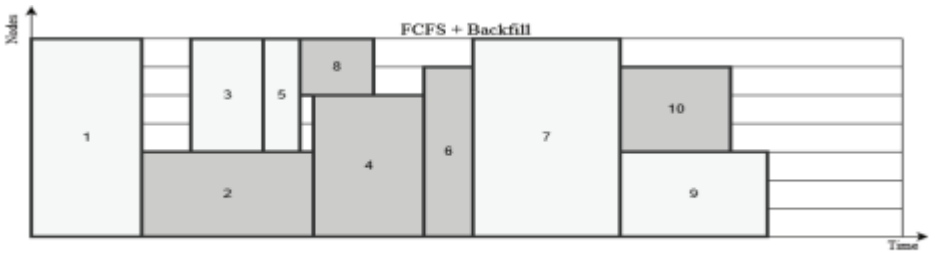


Fig. 1. Examples of FCFS and FCFS + Backfill.

Leonenkov and Zhumatiy (2015)
Introducing New Backfill-based Scheduler for SLURM Resource Manager

Resource Accounting and Limits

Swarm

Disk space:

- /home (10GB)
- /work1 (2TB)

User limits:

- 2240/2800 CPU limit
- ~~5.0~~1.0/6.4 TB RAM limit
- 10GB/allocated core (10GB/2 CPU)

Remember these rules

- DO NOT run anything on the head node --- always use srun or sbatch for anything computationally intensive

- DO NOT overallocate **time**, **memory**, or **CPU**

- **CHECK** your own jobs

BE RESPECTFUL!

First, you need an account

- Step 1. Get an account by having your advisor email CSCF
- Step 2. Log in with your CICS account

```
$ ssh ksung@swarm2.cs.umass.edu
ksung@swarm2.cs.umass.edu's password: hunter2

Last login: Sat Feb  2 23:11:24 2019 from c-66-31-41-74.hsd1.ma.comcast.net
Welcome to Bright release         7.3

                                                  Based on CentOS Linux 7
                                                             ID: #000002

Use the following commands to adjust your environment:

'module avail'            - show available modules
'module add <module>'     - adds a module to your environment for this session
'module initadd <module>' - configure module to be loaded at every login

-------------------------------------------------------------------------------


[ksung@swarm2 ~]$
```

SLURM commands

sbatch --- run an sbatch formatted file (normal way to run something)

srun --- run a command with specified resources. If within an sbatch file, it must be less than or equal to sbatch allocation. By default, the sbatch allocation will be used

squeue --- look at all submitted jobs by all users

Let's get something running!

- Example can be found in my home directory:

/home/ksung/resources

/process_example

Note: copy the whole directory to your own home directory before testing

Let's get something running!

Goal: parallelize the file on the right

Method 1: Make it runnable with command line arguments

Method 2: Parallelize it with a python library

import data

computation function

run 100 times

gather and write result

```
1   from random import random
2   # baseline 8088k RSS
3   with open('data','r') as data_file:
4     data = data_file.readlines()
5     # 801988k RSS, 184MB filesize
6
7   def compute(x):
8     results_list = []
9     for i in range(int(1e7)):
10      v = data[(x+i)%len(data)]
11      result = v * v
12      results_list.append(result)
13    return sum(results_list)
14  # 1.86 sec baseline
15  all_results = []
16  for _ in range(100):
17    all_results = compute(int(rand()*1e7))
18  # 10.31 sec for 1 run / 1119096k RSS
19  with open('results','w') as results_file:
20    results_file.writelines(
21      [str(r) for r in all_results]
22    )
```

process_serial.py

Let's get something running!

generate.py (generate example data to work with
--- shown here for replicability)

```
1   from random import random
2
3   def generate_data_file():
4     with open('data','w') as data_file:
5       for _ in range(int(1e7)):
6         data_file.write(str(random())+'\n')
7
8   generate_data_file()
```

```
[ksung@swarm2 swarmtest]$ srun python generate.py
[ksung@swarm2 swarmtest]$ ls -altrh data
-rw-r--r--+ 1 ksung brian 184M Feb  1 08:44 data
[ksung@swarm2 swarmtest]$ head data
0.68403579559481 66
0.72586166890301 84
0.84206187918535 86
0.18690552998501 92
0.65643917747785 81
0.744592432297663
0.35764222773784 45
0.5397177105307
0.96558193811477 82
0.90319879706280 81
```

Let's get something running --- profiling

process_serial.py

```
1    from random import random
2
```

Use srun and time to test and profile the script

```
[ksung@swarm2 swarmtest]$ srun time python process.py
0.02user 0.00system 0:00.04elapsed 90%CPU (0avgtext+0avgdata
 8088maxresident)k
0inputs+0outputs (0major+2125minor)pagefaults 0swaps
```

0.04 sec runtime at
90% CPU

8M memory

Let's get something running!

process_serial.py

```
1   from random import random
2
3   with open('data','r') as data_file:
4     data = data_file.readlines()
5
```

```
[ksung@swarm2 swarmtest]$ srun time python process.py
1.20user 0.50system 0:01.75elapsed 97%CPU (0avgtext+0avgdata
 802704maxresident)k
376376inputs+0outputs (0major+200857minor)pagefaults 0swaps
```

1.75 sec runtime at
97% CPU

803M memory

```python
from random import random
# baseline 8088k RSS
with open('data','r') as data_file:
    data = data_file.readlines()
    # 801988k RSS, 184MB filesize

def compute(x):
    results_list = []
    for i in range(int(1e7)):
        v = data[(x+i)%len(data)]
        result = v * v
        results_list.append(result)
    return sum(results_list)
# 1.86 sec baseline
all_results = []
for _ in range(100):
    all_results = compute(int(rand()*1e7))
# 10.31 sec for 1 run / 1119096k RSS
with open('results','w') as results_file:
    results_file.writelines(
        [str(r) for r in all_results]
    )
```

```
[ksung@swarm2 swarmtest]$ srun time python serial_process.py

9.07user 1.07system 0:10.28elapsed 98%CPU (0avgtext+0avgdata
 1119096maxresident)k
404200inputs+8outputs (5major+704966minor)pagefaults 0swaps
```

10.28 sec runtime at
98% CPU

1.2G memory

process_serial.py

Method I --- command line args

```python
from random import random
# baseline 8088k RSS
with open('data','r') as data_file:
    data = data_file.readlines()
    # 801988k RSS, 184MB filesize

def compute(x):
    results_list = []
    for i in range(int(1e7)):
        v = data[(x+i)%len(data)]
        result = v * v
        results_list.append(result)
    return sum(results_list)
# 1.86 sec baseline
all_results = []
for _ in range(100):
    all_results = compute(int(rand()*1e7))
# 10.31 sec for 1 run / 1119096k RSS
with open('results','w') as results_file:
    results_file.writelines(
        [str(r) for r in all_results]
    )
```

process_serial.py

```python
from random import random
import random,sys
run_number = sys.argv[1]
with open('data','r') as data_file:
    data = data_file.readlines()

def compute(x): # profile this
    results_list = []
    for i in range(int(1e7)):
        v = data[(x+i)%len(data)]
        result = v * v
        results_list.append(result)
    return sum(results_list)

with open('results'+str(run_number),'w') as results_f:
    results_f.write(
        str(compute(int(rand()*1e7)))+'\n'
    )
```

process_cmd.py

Anatomy of an sbatch file

```
#!/bin/bash
#SBATCH -j process_test  # name
#SBATCH -N 1             # number of nodes
#SBATCH -n 1             # number of tasks
#SBATCH -c 2             # number of cpus per task
#SBATCH --mem=1G         # memory per node
#SBATCH --mem-per-cpu=1G # memory per cpu
#SBATCH -a 0-99          # array
#SBATCH -t 00:01         # time allocated
#SBATCH -e process_test.err # error output file
#SBATCH -o process_out.out  # stdout file

srun process.py ${SLURM_ARRAY_TASK_ID}
```

More info: https://slurm.schedmd.com/sbatch.html

run.sb

```
#!/bin/bash
#SBATCH -j process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 2
#SBATCH --mem=1G
#SBATCH -a 0-99
#SBATCH -e process_test.err
#SBATCH -o process_out.out

srun process.py ${SLURM_ARRAY_TASK_ID}
```

process_cmd.py

```
from random import random
import random,sys
run_number = sys.argv[1]
with open('data','r') as data_file:
  data = data_file.readlines()

def compute(x): # profile this
  results_list = []
  for i in range(int(1e7)):
    v = data[(x+i)%len(data)]
    result = v * v
    results_list.append(result)
  return sum(results_list)

with open('results'+str(run_number),'w') as results_f:
  results_f.write(
    str(compute(int(rand()*1e7)))+'\n'
  )
```

run.sb

```
#!/bin/bash
#SBATCH -j process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 2
#SBATCH --mem=1G
#SE
#SE
#SE

sru
```

process_cmd.py

```
from random import random
import random,sys
run_number = sys.argv[1]
with open('data','r') as data_file:
  data = data_file.readlines()

def compute(x): # profile this
```

```
[ksung@swarm2 swarmtest]$ sbatch run.sb
Submitted batch job 9817176
[ksung@swarm2 swarmtest]$ squeue -u ksung
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         9817176_0      defq process_    ksung  R       0:04      1 swarm002
         9817176_1      defq process_    ksung  R       0:04      1 swarm008
[ksung@swarm2 swarmtest]$ ~/sueff.py ksung
user                 cpu eff         mem eff          alloc cpu       alloc mem
-------------------------------------------------------------------------------
ksung                49.10%          ≤ 0%             2               1G
```

```
  str(compute(int(rand()*1e7)))+'\n'
 )
```

Post-hoc profiling

run.sb

```
#!/bin/bash
#SBATCH -j process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 2
#SBATCH --mem=1G
#SBATCH
#SBATCH
#SBATCH

srun pro
```

```
[ksung@swarm2 swarmtest]$ sacct -j 9817176 -o Job,MaxRSS,TotalCPU,CPUTime,Elapsed
       JobID     MaxRSS   TotalCPU    CPUTime    Elapsed
------------ ---------- ---------- ---------- ----------
9817176_1                 00:10.266   00:00:22   00:00:11
9817176_1.b+       726K  00:00.013   00:00:22   00:00:11
9817176_1.0        582K  00:10.252   00:00:20   00:00:10
9817176_0                 00:11.203   00:00:24   00:00:12
9817176_0.b+       735K  00:00.014   00:00:24   00:00:12
9817176_0.0        588K  00:11.189   00:00:22   00:00:11
```

process_cmd.py

```
from random import random
import random,sys
run_number = sys.argv[1]
with open('data','r') as data_file:
  data = data_file.readlines()

def compute(x): # profile this
with open('results'+str(run_number),'w') as results_f:
  results_f.write(
    str(compute(int(rand()*1e7)))+'\n'
  )
```

Post-hoc profiling (throttled result)

run.sb

```
#!/b
#SBATCH -j process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 2
#SBATCH --mem=1G
#SBATCH
#SBATCH
#SBATCH

srun proc
```

process_cmd.py

```
from random import random
run_number = sys.argv[1]
with open('data','r') as data_file:
  data = data_file.readlines()

def compute(x): # profile this

with open( results  +str(run_number), w ) as results_f:
  results_f.write(
    str(compute(int(rand()*1e7)))+'\n'
  )
```

```
[ksung@swarm2 swarmtest]$ sacct -j 9817188 -o Job,MaxRSS,TotalCPU,CPUTime,Elapsed
       JobID     MaxRSS   TotalCPU    CPUTime    Elapsed
------------ ---------- ---------- ---------- ----------
9817188_1               00:09.913   00:02:20   00:01:10
9817188_1.b+      4696K 00:00.014   00:02:20   00:01:10
9817188_1.0     801932K 00:09.898   00:02:20   00:01:10
9817188_0               00:10.985   00:02:22   00:01:11
9817188_0.b+      4754K 00:00.014   00:02:22   00:01:11
9817188_0.0     800272K 00:10.971   00:02:22   00:01:11
```

Hyperthreading

- Non-MKL benchmark

-n 1

```
[ksung@swarm2 ~]$ sacct -j 9826135 -o
MaxRSS,TotalCPU,CPUTime,Elapsed
    MaxRSS   TotalCPU    CPUTime    Elapsed
---------- ---------- ---------- ----------
            00:09.680   00:00:26   00:00:13
      729K  00:09.680   00:00:26   00:00:13
```

-n 2

```
[ksung@swarm2 ~]$ sacct -j 9826136 -o
MaxRSS,TotalCPU,CPUTime,Elapsed
    MaxRSS   TotalCPU    CPUTime    Elapsed
---------- ---------- ---------- ----------
            00:35.875   00:00:38   00:00:19
      729K  00:35.875   00:00:38   00:00:19
```

- MKL benchmark

-n 1

```
[ksung@swarm2 benches]$ sacct -j 9825685 -o
MaxRSS,CPUTime,TotalCPU,Elapsed
    MaxRSS    CPUTime   TotalCPU    Elapsed
---------- ---------- ---------- ----------
            00:03:06  03:00.324   00:01:33
     5502K  00:03:06  00:00.014   00:01:33
   105819K  00:01:32  03:00.310   00:01:32
```

-n 2

```
[ksung@swarm2 benches]$ sacct -j 9825304 -o
MaxRSS,CPUTime,TotalCPU,Elapsed
    MaxRSS    CPUTime   TotalCPU    Elapsed
---------- ---------- ---------- ----------
            00:07:46  07:42.980   00:03:53
     4725K  00:07:46  00:00.014   00:03:53
   119065K  00:07:44  07:42.965   00:03:52
```

Hyperthreading

- Users can only book one whole core at a time (two threads with hyperthreading)
- Forcing your program to use both threads will probably not significantly increase your efficiency. It will however look like you're using only 50% of CPU
- Take advantage of libraries (like numpy) that optimize for hyperthreads! Python on swarm is compiled with Intel MKL support for hyperthreading. Anaconda's release should come with it, too.

Using a library is usually better

Pros

- Don't reinvent the wheel
- Can save memory and time
- Can consolidate (reduce) results more easily

Cons

- Libraries are language dependent
- It is sometimes harder to implement

Method 2 --- multiprocessing library

```
# baseline 8088k RSS
with open('data','r') as data_file:
  data = data_file.readlines()
  # 801988k RSS, 184MB filesize

def compute(x):
  results_list = []
  for i in range(int(1e7)):
    v = data[(x+i)%len(data)]
    result = v * v
    results_list.append(result)
  return sum(results_list)
# 1.86 sec baseline
all_results = []
for _ in range(100):
  all_results = compute(int(rand()*1e7))
# 10.31 sec for 1 run / 1119096k RSS
with open('results','w') as results_file:
  results_file.writelines(
    [str(r) for r in all_results]
  )
```

process_multi.py

```
from random import random
from multiprocessing import Pool

with open('data','r') as data_file:
  data = data_file.readlines()

def compute(x): # profile this
  results_list = []
  for i in range(int(1e7)):
    v = data[(x+i)%len(data)]
    result = v * v
    results_list.append(result)
  return sum(results_list)

p = Pool(20)
all_results = p.map(
  compute,
  [int(random()*1e7) for _ in range(100)]
)

with open('results','w') as results_file:
  results_file.writelines(
    [str(r) for r in all_results]
  )
```

run.sb

```
#!/bin/bash
#SBATCH -J process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 8
#SBATCH --mem=2G
#SBATCH -a 0
#SBATCH -e process_test.err
#SBATCH -o process_out.out

srun -c 8 python process_multi.py
```

(throttled result)

```
[ksung@swarm2 swarmtest]$ sacct -j 9838632 -o Job,MaxRSS,TotalCPU,
CPUTime,Elapsed
       JobID     MaxRSS   TotalCPU    CPUTime    Elapsed
------------ ---------- ---------- ---------- ----------
9838632_0                02:56.880   00:11:20   00:01:25
9838632_0.b+      4921K  00:00.015   00:11:20   00:01:25
9838632_0.0     800967K  02:56.865   00:11:20   00:01:25
```

Save memory with multiprocessing!

process_multi.py

```
1  from random import random
2  from multiprocessing import Pool
3
4  with open('data','r') as data_file:
5    data = data_file.readlines()
6
7  def compute(x): # profile this
8    results_list = []
9    for i in range(int(1e7)):
10     v = data[(x+i)%len(data)]
11     result = v * v
12     results_list.append(result)
13   return sum(results_list)
14
15 p = Pool(20)
16 all_results = p.map(
17   compute,
18   [int(random()*1e7) for _ in range(100)]
19 )
20
21 with open('results','w') as results_file:
22   results_file.writelines(
23     [str(r) for r in all_results]
24   )
```

run.sb

```
#!/bin/bash
#SBATCH -J process_test
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 8
#SBATCH --mem=2G
#SBATCH -a 0
#SBATCH -e process_test.err
#SBATCH -o process_out.out

srun -c 8 python process_multi.py
```

(throttled result)

```
[ksung@swarm2 swarmtest]$ sacct -j 9838647 -o Job,MaxRSS,TotalCPU,
CPUTime,Elapsed
       JobID     MaxRSS   TotalCPU    CPUTime    Elapsed
------------ ---------- ---------- ---------- ----------
9838647_0                06:27.126   00:14:56   00:01:52
9838647_0.b+      4779K  00:00.016   00:14:56   00:01:52
9838647_0.0    8991875K  06:27.110   00:14:56   00:01:52
```

Copy-on-write causes 8x memory usage

process_multi.py

```
1   from random import random
2   from multiprocessing import Pool
3
4   with open('data','r') as data_file:
5     data = data_file.readlines()
6
7   def compute(x): # profile this
8     results_list = []
9     for i in range(int(1e7)):
10      v = data[(x+i)%len(data)]
11      data[(x+i)%len(data)] = 0
12      result = v * v
13      results_list.append(result)
14    return sum(results_list)
15
16  p = Pool(20)
17  all_results = p.map(
18    compute,
19    [int(random()*1e7) for _ in range(100)]
20  )
21
22  with open('results','w') as results_file:
23    results_file.writelines(
24      [str(r) for r in all_results]
25    )
```

Don’t commit these sins

- DO NOT run anything on the head node --- always use srun or sbatch for anything computationally intensive

- DO NOT overallocate **time**, **memory**, or **CPU**

- **CHECK** your own jobs

BE RESPECTFUL!

Other tips

- Minimize reads and writes to disk
- Write fault-tolerant code
  - Save “state” often so that code can restart if it fails for any reason
- Make your program as fragmentable as possible. It is easier to schedule a high number of low resource jobs than a lower number of resource intensive jobs

Checkpointing with DMTCP

- Example can be found in my home directory:

/home/ksung/resources

/dmtcp_example

Note: copy the whole directory to your own home directory before testing

Checkpointing with DMTCP (experimental)

- Any job with more than one node will be buggy
- Saves memory state to filesystem

Start a job: `sbatch slurm_launch.job`

Continue a job: `sbatch slurm_rstr.job`

- /home/ksung/dmtcp_example

```
-rw-------+  1 ksung brian  29M Feb  2 23:38 ckpt_python3.5_471891ed5a75b2e2-45000-5b200cda64dd4.dmtcp
-rw-------+  1 ksung brian  29M Feb  2 23:38 ckpt_python3.5_471891ed5a75b2e3-40000-5b203b8d6e72c.dmtcp
-rw-------+  1 ksung brian  29M Feb  2 23:38 ckpt_python3.5_471891ed5a75b2e3-42000-5b203b8d74974.dmtcp
-rw-------+  1 ksung brian  29M Feb  2 23:38 ckpt_python3.5_471891ed5a75b2e3-43000-5b203b8c96a52.dmtcp
-rwxr--r--+  1 ksung brian 6.8K Feb  2 23:38 dmtcp_restart_script_471891ed5a75b2e3-40000-5b203b7babb47.sh
lrwxrwxrwx   1 ksung brian   60 Feb  2 23:38 dmtcp_restart_script.sh -> dmtcp_restart_script_471891ed5a75b2e3-40000-5b203b7babb47.sh
```

DMTCP

slurm_launch.out

```
[ksung@swarm2 dmtcp2]$ tail -30 dmtcp.out
swarm001 19
swarm001 19
swarm001 19
swarm002 19
swarm001 20
swarm001 20
swarm001 20
swarm002 20
swarm001 21
swarm001 21
swarm001 21
swarm002 21
swarm001 22
swarm001 22
swarm001 22
swarm002 22
swarm001 23
swarm001 23
swarm001 23
swarm002 23
swarm001 24
swarm001 24
swarm001 24
swarm002 24
slurmstepd: error: Step 9829757.0 exceeded memory limit (126337 > 102400), being killed
slurmstepd: error: *** STEP 9829757.0 ON swarm001 CANCELLED AT 2019-02-02T23:35:55 ***
slurmstepd: error: Exceeded job memory limit
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
srun: error: swarm002: task 7: Killed
srun: error: swarm001: tasks 0-1,3: Killed
```

slurm_rstr.out

```
swarm001 20
swarm001 20
swarm001 20
swarm001 20
swarm001 21
swarm001 21
swarm001 21
swarm001 21
swarm001 22
swarm001 22
swarm001 22
swarm001 22
swarm001 23
swarm001 23
swarm001 23
swarm001 23
swarm001 24
swarm001 24
swarm001 24
swarm001 24
swarm001 25
swarm001 25
swarm001 25
swarm001 25
swarm001 26
swarm001 26
swarm001 26
swarm001 26
swarm001 27
swarm001 27
swarm001 27
swarm001 27
```

DMTCP

Excerpt from slurm_launch.job

```
#######################################################################
# 1. Start DMTCP coordinator
#######################################################################

start_coordinator -i 10 # ... <put dmtcp coordinator options here>

#######################################################################
# 2. Launch application
# 2.1. If you use mpiexec/mpirun to launch an application, use the following
#      command line:
#        $ dmtcp_launch --rm mpiexec <mpi-options> ./<app-binary> <app-options>
# 2.2. If you use PMI1 to launch an application, use the following command line:
#        $ srun dmtcp_launch --rm ./<app-binary> <app-options>
# Note: PMI2 is not supported yet.
# 2.3. If you use the Stampede supercomputer at Texas Advanced Computing Center
#      (TACC), use ibrun command to launch the application (--rm is not required):
#        $ ibrun dmtcp_launch ./<app-binary> <app-options>
#######################################################################

# dmtcp_launch --rm mpirun --mca btl self,tcp ./<your binary>
srun dmtcp_launch --rm python count.py
```

Excerpt from slurm_rstr.job

```
#######################################################################
# 1. Start DMTCP coordinator
#######################################################################

start_coordinator # -i 120 ... <put dmtcp coordinator options here>

#######################################################################
# 2. Restart application
#######################################################################

/bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORT

#######################################################################
# If you use the Stampede supercomputer at Texas Advanced Computing Center
# (TACC), add the --hostfile option:
# /bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORT\
#                                     --hostfile $HOSTFILE
#######################################################################
```

Troubleshooting

- Memory error

```
slurmstepd: error: Step 9829757.0 exceeded memory limit
(126337 > 102400), being killed
slurmstepd: error: *** STEP 9829757.0 ON swarm001
CANCELLED AT 2019-02-02T23:35:55 ***
slurmstepd: error: Exceeded job memory limit
srun: Job step aborted: Waiting up to 32 seconds for job
step to finish.
srun: error: swarm002: task 7: Killed
srun: error: swarm001: tasks 0-1,3: Killed
```

Troubleshooting

- Time expiry error
  - SIGTERM 32 sec before SIGKILL

```
import signal
import sys
from time import sleep

def sigterm_handler(_signo, _stack_frame):
  print('sorry',flush=True)
  for i in range(1000):
    print(i,flush=True)
    sleep(1)
  sys.exit(0)

signal.signal(signal.SIGTERM, sigterm_handler)

sleep(600)
```

```
[ksung@swarm2 dmtcp2]$ srun -t 00:00:01
python term_test.py
srun: Force Terminated job 9844110
srun: Job step aborted: Waiting up to
32 seconds for job step to finish.
slurmstepd: error: *** STEP 9844110.0
ON swarm002 CANCELLED AT 2019-02-
05T14:07:34 DUE TO TIME LIMIT ***
sorry
0
.
.
.
28
29
srun: error: swarm002: task 0: Killed
```

Troubleshooting

- Allocation error --- your allocation doesn't make sense
- Assoc Limit --- you or your group is currently already maxing out your resource limit
- Resource --- you are first in line but there are not enough resources for your job
- Priority --- you are waiting for the first in line (Resource) to be scheduled

Troubleshooting

```
[ksung@swarm2 swarmtest]$ ~/blame.py 1

Percent of resources allocated per user
                 alloc            wasted
user             cpu      mem     cpu      mem     num_nodes      cpus_per_node    mem_per_node
                 6.9%    12.1%    2.0%    10.8%           15               14.1            61.0G
                 4.2%    13.5%    1.9%     3.8%           14                9.1            73.1G
                 1.0%     1.2%    0.8%     1.0%            1               30.0            93.0G
                 1.0%     0.7%    0.8%     0.6%            3               10.0            16.7G
                 0.6%     0.1%    0.3%     0.1%            5                3.6             2.2G
                 9.4%     3.8%    0.1%     1.0%            8               36.0            36.0G
                 3.2%     2.6%    0.0%    -0.6%            2               50.0           100.0G

Total swarm allocation

cpu      mem
0.26     0.34


Resources available

cpu      mem
2274     4989.2G

Conflicts:
REASON  PRIORITY        CPUS    NODES   MIN_MEMORY      USER
Resources       5004    30      1       93000M                  x1

Waiting:
REASON  PRIORITY        CPUS    NODES   MIN_MEMORY      USER
ReqNodeNotAvail  UnavailableNod 5004    30      1       93000M                  x9
AssocGrpMemLimit        4150    1       1       16000M          x1697
JobArrayTaskLimit       4004    4       1       15000M                  x1
```

/home/ksung/blame.py

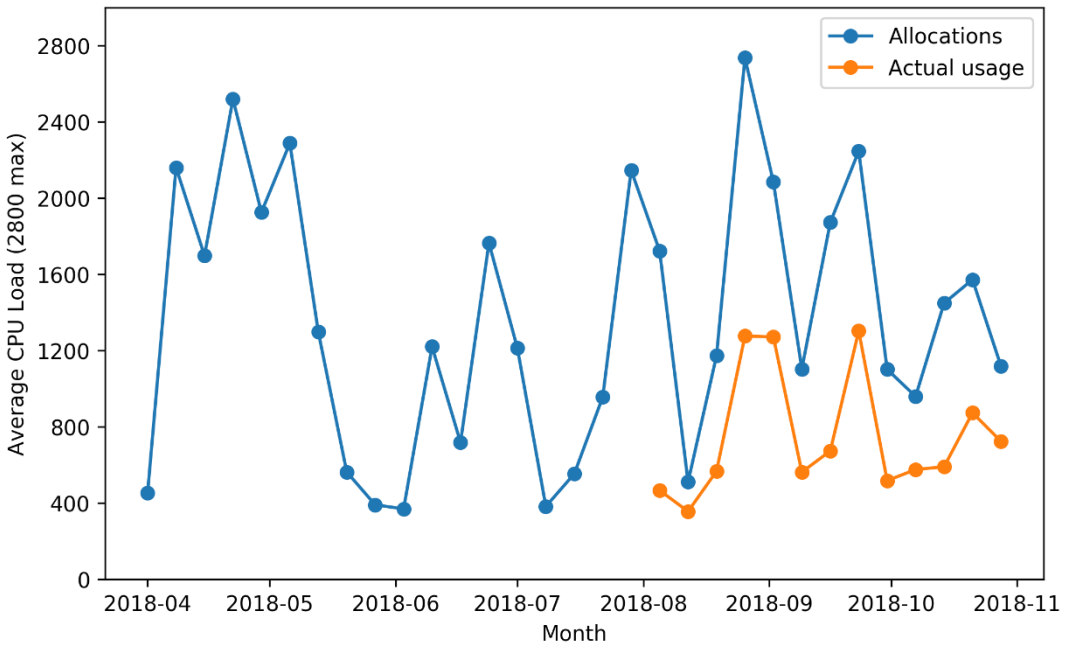
```
[ksung@swarm2 ~]$ blame.py

Percent of resources allocated per user
                 alloc            wasted
user             cpu      mem     cpu      mem     num_nodes      cpus_per_node    mem_per_node
[redacted]       7.1%     12.5%   2.2%     11.2%           15               14.7            63.0G
                 4.2%     13.5%   1.9%     5.6%            14               9.1             73.1G
                 1.9%     1.3%    1.7%     1.3%            6                10.0            16.7G
                 1.0%     1.2%    0.8%     1.0%            1                30.0            93.0G
                 0.6%     0.1%    0.3%     0.1%            5                3.6             2.2G
                 0.1%     0.1%    0.1%     0.1%            1                2.0             4.0G
                 9.4%     3.8%    0.1%     1.0%            8                36.0            36.0G
                 3.2%     2.6%    0.0%     -0.6%           2                50.0            100.0G

Total swarm allocation

cpu      mem
0.27     0.35


Resources available

cpu      mem
2234     4905.2G
```

/home/ksung/sueff.py

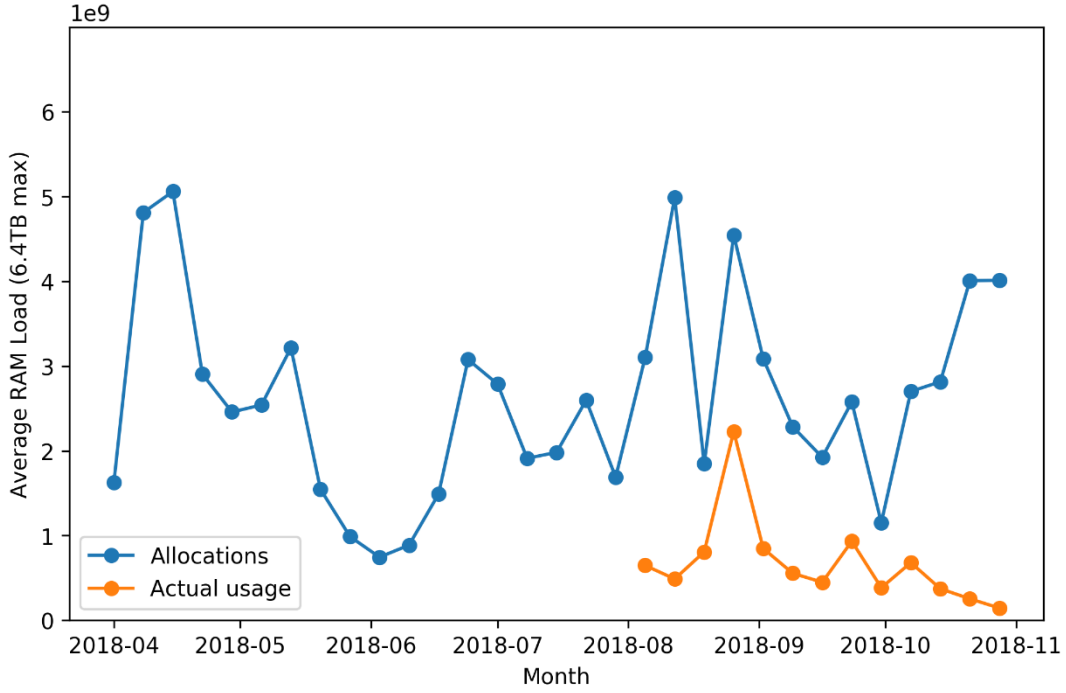
```
[ksung@swarm2 ~]$ sueff.py [redacted]
user                 cpu eff         mem eff          alloc cpu       alloc mem
-------------------------------------------------------------------------------
[redacted]           69.70%          ≤ 10%            220             945G
```

Usage history

CPU (29%/50%)



Memory (11%/41%)



Policy changes to expect in the near future

Motivation: increase swarm efficiency, use, fairness, and turnover

- Shorter defq time and more defq-only nodes
- Changes in fairness calculation

Commands you should use often

squeue -u <user>

sbatch <sbatch file>

srun time <executable>

sacct -j <JobID> -o Job,MaxRSS,TotalCPU,CPUTime,Elapsed

blame (/home/ksung/resources/bin/blame)

sueff (/home/ksung/resources/bin/sueff)

List of resources

- /home/ksung/resources/install --- install dmtcp, sueff, and blame

Examples:

- /home/ksung/resources/dmtcp_example
- /home/ksung/resources/process_example

https://slurm.schedmd.com/sbatch.html

https://people.cs.umass.edu/~swarm/index.php?n=Main.NewSwarmDoc

Summary

- DO NOT run anything on the head node --- always use srun or sbatch for anything computationally intensive

- Profile your program!
  - DO NOT overallocate **time**, **memory**, or **CPU**

- **CHECK** your own jobs when you run them

BE RESPECTFUL!

Install the tools:
```
$ /home/ksung/resources/install
```

Monitor the mailing list:
swarm-users@cs.umass.edu

Issues?
Email the mailing list or Keen: ksung@cs.umass.edu