MANAGING YOUR MEMORY FOOTPRINT on the Discover cluster

NCCS lunchtime series

May 27, 2014

another one of many lunchtime talks...

György (George) Fekete
Why is memory management important?

Possible consequences of treating memory poorly...

Dear user,

From the information supplied by the Discover Cluster, your job, 1234567, incurred 69.600% swapping during its processing and came dangerously close to running one or more nodes out of memory. In order to avoid a GPFS hang this job was terminated by the system. We encourage users whose jobs are terminated in this manner to contact the NCCS User Services Group for assistance in identifying the underlying cause of the high swapping.

We apologize for any inconvenience this situation may have caused you...

Entered on XX/XX/XXXX at XX:XX:XX EDT (GMT-0400) by Michael (Mike) W. Donovan:

Tue May 20 09:49:57 EDT 2014 borg01t123 was REBOOTED. (user) (1234567) (69.600 % swap)
Tue May 20 09:50:11 EDT 2014 borg01t138 was REBOOTED. (user) (1234567) (69.200 % swap)
Tue May 20 09:50:22 EDT 2014 borg01t124 was REBOOTED. (user) (1234567) (69.100 % swap)
Why is memory management important?

Possible consequences of treating memory poorly...

Dear user,

From the information supplied by the Discover Cluster, your job, 1234567, incurred **69.600% swapping** during its processing and came dangerously close to running one or more nodes out of memory. In order to **avoid a GPFS hang** this job was terminated by the system. We encourage users whose jobs are terminated in this manner to contact the NCCS User Services Group for assistance in identifying the underlying cause of the high swapping.

We apologize for any inconvenience this situation may have caused you...

Entered on XX/XX/XXXX at XX:XX:XX EDT (GMT-0400) by Michael (Mike) W. Donovan:

Tue May 20 09:49:57 EDT 2014 borg01t123 was **REBOOTED**. (user) (1234567) (69.600 % swap)

Tue May 20 09:50:11 EDT 2014 borg01t138 was **REBOOTED**. (user) (1234567) (69.200 % swap)

Tue May 20 09:50:22 EDT 2014 borg01t124 was **REBOOTED**. (user) (1234567) (69.100 % swap)
What's wrong with that?

• Why was the job terminated?
  • Swap ratio was too high

• Why do we care? It was nowhere near 100% -- yet
  • If it reaches 100% it may be too late to do anything about it
  • GPFS hangs are bad for your node and it will take down others with it, which, in turn, take down more, and so on

• Why is good to stop this?
  • A node rebooted is unavailable for at least 10 minutes
    • On weekends and nights could be more
  • Waste of time and SBUs and jobs must be submitted again
Today's topics

• How to avoid running nodes out of memory
  • assess memory requirements accurately
  • monitor memory profile
  • jump ship if memory consumption goes out of control
  • distribute and balance tasks
    • use as few nodes as possible
    • use as many nodes as necessary

• Some useful terms
  • swapping
  • peak resident set size
Swapping

- Virtual memory, arranged in *pages*
  - physical (RAM, memory)
  - disk
- Page fault
  - attempt to access memory currently not in RAM
  - page in RAM is *swapped* for page on disk
- Big data space in big program
  - many page faults
  - too much swapping a.k.a *thrashing*
- *At some point* …
Swapping (cont'd.)

• **At some point computer spends more of its resources to manage swapping than useful computation**
  • It is vitally important to keep swapping to a minimum

• Manage memory well -- try to use only physical memory
  • Assess memory requirements
  • Distribute work load sensibly
    • Some nodes will use more cores than others
  • Understand how computer uses memory
    • *Be aware of not so obvious caveats*
Why is memory running out?

• Common misconceptions

  • This node has 32 GB of memory, surely that's enough!
    – Actually, 2GB is reserved for system use, so only 30GB left.
    – Things that can affect total memory consumption:
      » using more than one CPU on this node?
      » running more than one process on this node?

  • Virtual memory is limited only by disk space.
    – Sort of -- limited to disk space configured for this purpose.
    – Irrelevant, use physical memory

  • Array allocations were successful, there were no errors, so doesn't the process have complete ownership of allocated memory?
    – Yes and no
Today's topic in detail

1. Watch your environment
   - physical memory in use
   - swap ratio
   - your process's memory high water mark

2. Deal well with *embarrassingly parallel tasks* (e.g. GrADS plots)
   - independent plots are generated/specified by
     » list of parameters
     » a list of fully qualified commands

3. Force an arbitrary *(uneven)* load distribution in an MPI job
   - MPI ranks $i$, $j$, $k$... require more memory than other ranks
     - Use more cores per node: get less memory per task
     - Use fewer cores per node: get more memory per task
Watch your environment (topic 1)

- Physical memory (RAM) in use
  - for smooth operation < 96%
  - if you allocate and not free storage this number will grow
  - when this number exceeds 96% expect swapping to start

- Swap ratio
  - used swap space / total swap space
  - anything over a few % is bad
  - when this number exceeds 60% expect termination

Global, per node properties
Watch your environment

- **High water mark**
  - Per process peak *resident set size*
  - VmHWM in /proc/$PID/status

---

http://locklessinc.com/articles/memory_usage/
Resident set size vs. Peak resident set size

Memory Usage

MIB

0 100 200 300

time (s)
How fast can a node run out of memory?

• **Physical**
  • very fast, a couple of seconds

• **Virtual**
  • Swapping slows everything down, therefore the rate at which virtual memory is exhausted is also slower.

• **Useful metric**
  • Since any node that exceeds 60% swap ratio is rebooted, it is useful to study the rate at which the swap ratio changes
Why does a node run out of memory?

- Linux memory allocation with an example
  - process 1
    - check how much free (physical) memory is available
      - note: memory is shared by all CPUs on a node
    - successfully allocate 10% of it
  - process 2,3,…, 16
    - same…
  - 16 * 10% > 100%!
- allocation is more like a reservation
  - process has it when it deposits something in it
  - most programs on a single thread don't run into trouble
  - but others can!
Running into trouble

• Example:
  • MPI program on 1 node, 16 tasks per node
    • use all 16 CPUs on Sandy Bridge node
  • Each rank:
    • successfully allocates a large chunk of memory
    • immediately starts filling it
  • Observe:
    • monitor peak resident memory (high watermark)
Peak resident total memory (high water mark)

exhaust physical memory in 2 seconds
Swap ratio time profile

% swap ratio
Bad news: start swapping as soon as physical memory is exhausted

Good news: rate of increase in swap ratio is limited

This is absolute worst case scenario this test was trying to run a node out of memory deliberately
Simple NCCS tools to monitor memory usage

- Add lines to start *policeme* to job script
  - decide if a preemptive signal at a target swap ratio is needed
    - select the swap ratio at which signal will be sent
    - must be < 60%, else swapkiller takes drastic measures
  - select the frequency with which the monitor assesses the situation
    - number of seconds between observations (*timesteps*)
  - SLURM job file will require only a few extra lines
Memory police

• Report for every timestep:
  • peak resident memory for each task
  • total peak resident memory for each node
  • swap ratio for each node

• *if you want others, we can provide*...
Sample SLURM batch script

```bash
#!/bin/bash
#SBATCH --job-name=swapkiller --time=00:05:00
#SBATCH --nodes=1 --ntasks-per-node=16
#SBATCH --output=out.txt
#SBATCH --error=err.txt

. /usr/share/modules/init/bash
module purge
module load comp/intel-13.0.1.117 mpi/impi-4.0.3.008

export MEMTOOLS=/usr/local/other/primertools
source $MEMTOOLS/lib/policeme.inc

trap "{
    echo swapkiller or something is terminating me;
}" SIGTERM

policeme 230 1 55 mpiprogs `pwd`
mpirun $SOMEPATH/mpiprog
wait `cat scpid`
```

Comment
scpid is a file that keeps the process ID for the watchdog
#!/bin/bash
#SBATCH --job-name=swapkiller --time=00:05:00
#SBATCH --nodes=1 --ntasks-per-node=16
#SBATCH --output=out.txt
#SBATCH --error=err.txt

. /usr/share/modules/init/bash
module purge
module load comp/intel-13.0.1.117 mpi/impi-4.0.3.008

export MEMTOOLS=/usr/local/other/primertools
source $MEMTOOLS/lib/policeme

trap "{
    echo swapkiller or something is terminating me;
    }" SIGTERM

policeme 230 1 55 mpiprog `pwd`
mpirun $SOMEPATH/mpiprog
wait `cat scpid`
Sample SLURM batch script

```
policeme 120 1 55 mpiprog `pwd`
```

- number of seconds
- time (s) between taking a look
- target swap ratio
- MPI program
- directory for output

- timestamp
- swap ratio
- free physical memory
- total used memory
- number of seconds
- percent
- kB
- permil
Format of output from memory police

<table>
<thead>
<tr>
<th>TS</th>
<th>SR</th>
<th>FR</th>
<th>US</th>
<th>Node</th>
<th>Processor</th>
<th>Processes</th>
<th>Memory Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.634001</td>
<td>30503484</td>
<td>71</td>
<td>borg01w020</td>
<td>0 55</td>
<td>1 1718</td>
<td>mpiprog 3308 kB</td>
</tr>
<tr>
<td>1</td>
<td>1.634001</td>
<td>30469688</td>
<td>72</td>
<td>borg01w020</td>
<td>0 55</td>
<td>1 1719</td>
<td>mpiprog 3292 kB</td>
</tr>
<tr>
<td>2</td>
<td>1.634001</td>
<td>30469316</td>
<td>72</td>
<td>borg01w020</td>
<td>0 55</td>
<td>1 1733</td>
<td>mpiprog 3292 kB</td>
</tr>
<tr>
<td>27</td>
<td>58.184774</td>
<td>171948</td>
<td>994</td>
<td>borg01w020</td>
<td>CRIT</td>
<td>27 1718</td>
<td>mpiprog 1905868 kB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>27 1719</td>
<td>mpiprog 1972372 kB</td>
</tr>
</tbody>
</table>

"policeme" will create a file for each node involved in your job
each file contains trace output for each process matching the given name
Summary of memory watchdog

• **Monitor memory usage**
  • per process or task/CPU
    • peak resident set size (high watermark)
  • per node
    • physical memory in use
    • swap ratio
    • free physical memory
Embarrassingly parallel tasks (topic 2)

- Many independent tasks
e.g. a few hundred GrADS plots
  *here we assume that all jobs have about the same memory requirements*

- Challenges
  - run them on the cluster in parallel
    - use as few nodes as possible, but...
    - use as many nodes as necessary
  - preparation
    - assess peak resident set size for a task *(see earlier)*
    - calculate how many tasks can fit on one node
      - Westmere: 24 GB - reserved = cca. 22GB
      - Sandy Bridge: 32 GB -reserved = cca. 30GB
Independent tasks on multiple nodes

• The story so far:
  • know everything to begin distributing tasks
  • use 2 nodes and force only 1 CPU per node
    » nodes=2, ntasks=2
  • commandlist ready
    » one line per command
    » arbitrary number of tasks

• Workflow treat commandlist as a TODO list
  – while commandlist is not empty
    » get next command and truncate list
    » make it run on an idle node
    » repeat
Implementation of "todo list" workflow

- Turn it inside out:
  - Each node runs a task that reads the list
  - Pure shell commands; very little overhead so you get more memory for your tasks
- Each task (usually one per CPU)
  - save first line from "todo list" into shell variable XQT
  - remove first line
  - execute XQT How does this stop?
    - if todo list does not exist then exit
    - if todo list is empty, delete todo list

Potential problems?
Implementation of "todo list" workflow (cont'd.)

Potential problems?

FACT:
There is only one todo list, but several jobs running independently of each other on multiple nodes/cores.

PROBLEMS:
It would be bad if
more than one process read the same line
more than one process tried to remove the top line at the same time

Creating temporary files is an unnecessary load that should be avoided.
Edit the todo list without creating temporary files.
while there is a \textit{todolist} \\
\textbf{enter critical section} \\
if there is still a \textit{todolist} \\
\hspace{2em} tear off the fist line in \textit{todolist} into \textbf{command} \\
\hspace{2em} let n = number of lines in \textit{todolist} \\
\hspace{2em} if \textit{todolist} is empty \\
\hspace{4em} remove \textit{todolist} \\
\textbf{exit critical section} \\
run the \textbf{command} \\
\textbf{exit critical section} \\
end while
Patterns used in implementation

% lockfile -1 LOCKFILE

Used when entering a critical section. First call creates LOCKFILE and returns immediately. Subsequent calls block until LOCKFILE is removed. This is how we prevent from more than one process to be in the critical section. To exit the critical section, simply remove LOCKFILE with 'rm -f'

% set XQT = `head -1 todolist`

Read the first line of todolist into a shell variable

% sed 1,1d -i todolist

Remove the first line of todolist

% set n = `wc -l todolist | cut -d" " -f 1`

Put the number of lines in todolist into a shell variable. There are other ways to check whether or not a file is empty, but this pattern has other uses, like checking to see if a file has only one line in it, for example.
Put it all together *shell script*

```bash
#!/bin/csh -f

while ( -e todolist )
    lockfile -1 LOCKFILE
    if ( -e todolist ) then
        set XQT = `head -1 todolist`
        sed 1,1d -i todolist
    endif
    if ( -z todolist ) then
        /bin/rm -f todolist
    endif
    rm -f LOCKFILE
    ${XQT}
end

This is the only important line

All of this is the "performer" script
```
Put it all together (cont'd.)

```
$ sbatch driver.csh
```

driver.csh
```
#!/bin/csh -f
#SBATCH --time=01:00:00
#SBATCH --nodes=2
#SBATCH --ntasks=2
srun performer.csh
```

performer.csh
```
#!/bin/csh -f
while ( -e todolist )
  lockfile -1 LOCKFILE
  if ( -e todolist ) then
    set XQT = `head -1 todolist`
    sed 1,1d -i todolist
    set n = `wc -c todolist | cut -d" " -f 1`
    if ( $n == 0 ) then
      /bin/rm -f todolist
    endif
    /bin/rm -f LOCKFILE
    ${XQT}
  endif
  rm -f LOCKFILE
end
```
Unbalanced MPI loads (topic 3)

- Default even load balancing
  - ok, if all tasks have about the same memory footprints
  - not ok, if tasks have uneven memory requirements

<table>
<thead>
<tr>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>manages many tasks</td>
<td>special task</td>
<td>special task</td>
<td>special task</td>
</tr>
<tr>
<td>collects massive data</td>
<td>little memory</td>
<td>little memory</td>
<td>little memory</td>
</tr>
</tbody>
</table>

Even distribution does not work so well:
- either you waste a lot of cores
- or you squeeze too many tasks into one node and you run out of memory

You may dedicate one node to Rank 0 only
and have several of the lesser ranks share a node
Arbitrary distribution of tasks to nodes

- Arbitrary here means *any way you want*, override system defaults
- **SLURM/MPI**
  - Say how many nodes you want
  - Say how you want your MPI ranks distributed among the nodes

<table>
<thead>
<tr>
<th>MPI Rank</th>
<th>Node number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

create a `hostindex` file

<table>
<thead>
<tr>
<th>MPI rank</th>
<th>node index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
SLURM needs a "hostfile"

### How you want it

<table>
<thead>
<tr>
<th>MPI Rank</th>
<th>Node number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### Host index file

<table>
<thead>
<tr>
<th>MPI Rank</th>
<th>Node number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### How SLURM needs it: hostfile

- borg01w017
- borg01w018
- borg01w019
- borg01w019

Small problem: hostfile can not be created until jobs is scheduled and ready to run. Given: SLURM maintains names of allocated nodes in SLURM_NODELIST (shell var) solution: hi2n.py creates hostfile at runtime.

```
$SLURM_NODELIST
hostindex
```

```
hi2n.py
```
Sample SLURM batch script

#!/bin/bash

#SBATCH --job-name=uneven
#SBATCH --time=00:01:00
#SBATCH --nodes=3 --constraint=sand
...

umask 022
. /usr/share/modules/init/bash
module purge
module load comp/intel-13.1.3.192 mpi/impi-4.1.1.036

export I_MPI_USE_DYNAMIC_CONNECTIONS=0
export I_MPI_PMI_LIBRARY=/usr/slurm/lib64/libpmi.so

/usr/local/other/primertools/bin/hi2n.py < hostindex > hostfile
export SLURM_HOSTFILE=hostfile
srun -m arbitrary --ntasks=numberRanks myprogram.exe

exit 0
SUMMARY

• Know your memory usage
  • Use tools
    • proprietary profiling tools
    • in-house memory police
  • Be proactive
    • anticipate failure
    • avoid catastrophic failure with self-termination
    • use more nodes and fewer tasks per node
    • rebalance MPI ranks
Thank You!

- As always, feel free to contact the NCCS User Services Group with questions or problems
  - 301-286-9120
  - support@nccs.nasa.gov