Feb 22, 2013

Explanatory meeting for users of supercomputer system -- Knowhow for entering jobs in UGE --

Purpose of this lecture

To understand the method for smoothly executing a few hundreds or thousands of jobs

What is Univa Grid Engine (UGE)?

- It is software that is used to construct a grid computing system. It functions as a batch job system.
- It is a commercial product derived from Sun Grid Engine 6.2U5 (the last open-source version). The main developers of SGE participated in the development of UGE.
- The commands, etc., for entering jobs in the UGE are the same as that in SGE.

Advantages of using UGE

- Multiple jobs can be smoothly executed in a sequential manner.
- When more than one user enters several jobs simultaneously, UGE carries out scheduling.
- Effective scheduling is carried out according to the memory, CPU, etc., required by a job.

Precautions for using UGE

- Parallelization of jobs and other such functions cannot be performed.
- If the resource demand when a job is entered is not declared correctly, large-scale hang-up of computer may occur.

Contents

1. Before entering multiple jobs
2. Configuration/characteristics of supercomputer system
3. Finding the load property of job to be entered
4. Finding the method for managing the execution statuses of jobs

Before entering multiple jobs (1)

- If a few hundreds to thousands of jobs are entered simultaneously, the following points, which are typically not problematic, can become challenging:

- A small load per job increases if the number of simultaneous jobs increases. Pay attention to CPU load, memory consumption, and particularly I/O.
- If the load of a job is not known, large-scale problems such as a hang-up of hundred units or failure of the file system may occur.
- For a large number of jobs, a system that efficiently manages the jobs, such as checking of results, is required.

Before entering multiple jobs (2)

- Entering multiple jobs requires some preparation.
- If preparations are made appropriately, a job can be executed at a very fast rate.
- However, if this preparation is neglected, the job execution is delayed by several fold to dozens of times. Overload is a nuisance to other users, and it can stop the system in the worst case.

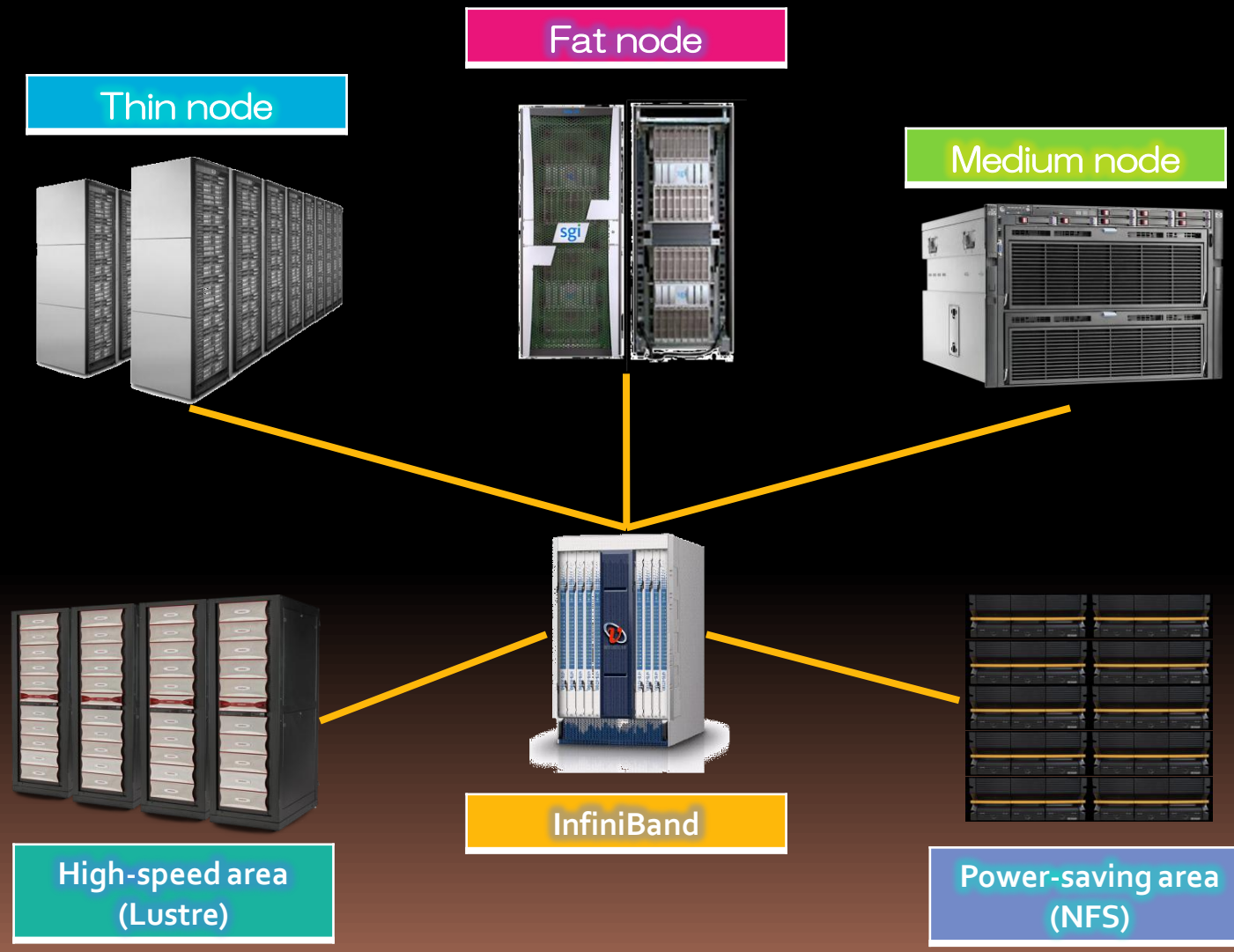
To enter multiple jobs

- Understand the configuration/characteristics of the system
- Find out the load property of any job (CPU, memory, and I/O)
- Find the method for managing job execution status

Contents

1. Before entering multiple jobs
2. Configuration/characteristics of supercomputer system
3. Finding the load property of job to be entered
4. Finding the method for managing the execution statuses of jobs

Overview of supercomputer system



Contents

2.1 Characteristics of each calculation node

2.2 Characteristics of file system

2.3 Upper limit value when a job is entered into UGE

Characteristics of each calculation node (1)

- Thin node (162 units)
- Thin node (SSD mounted) (76 units)
- Thin node (SSD, GPU mounted) (64 units)

- Amount of mounted memory: 64 GB
- CPU: Xeon E5-2670 (SandyBridge-EP) 2.6 GHz
  8 cores × 2 sockets (16 cores)
- SSD and GPU are mounted depending on the node

- The processing capacity per CPU core is the largest in these three types of nodes. This node should be actively used for processing when the required memory per job falls within the maximum amount of mounted memory (64 GB).

Characteristics of each calculation node (2)

- Medium node (2 units)

  - Amount of mounted memory: 2 TB
  - CPU: Xeon E7-4870 (Westmere EX) 2.4 GHz
    10 cores × 8 sockets (80 cores)

  - This node should be used for processing when the required memory per job is more than 64 GB and less than 2 TB.

- Fat node (1 unit)

  - Amount of mounted memory: 10 TB
  - CPU: Xeon E7-8837 (Westmere EX) 2.67 GHz
    8 cores × 96 sockets (768 cores)

  - This node should be used for processing when 2 TB or more shared memory is required.

Contents

2.1 Characteristics of each calculation node

2.2 Characteristics of file system

2.3 Upper limit value when job is entered into UGE

Overview of file system

- /lustre1 and /lustre2 compose a file sharing system, which can be referenced from all calculation nodes.

- The home directory of each user is distributed to /lustre1 and /lustre2.
- The location of your home directory can be checked using the command shown below:

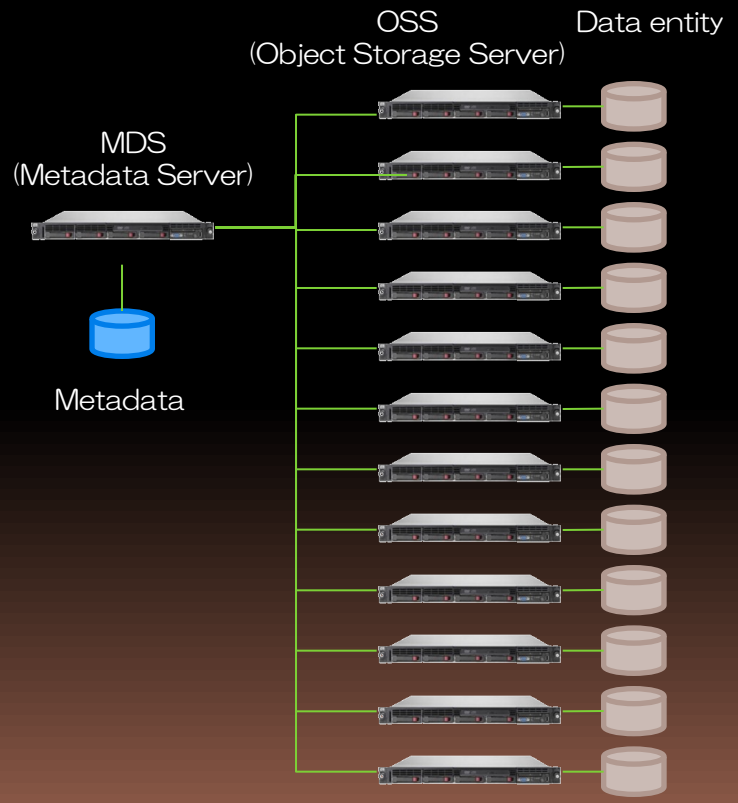
```
$ ls -l /home/lect01
lrwxrwxrwx 1 root root 20  March 18 15:35 2012 /home/lect01 ->
/lustre1/home/lect01
$
```

Details of file system

- /lustre1 and /lustre2 are configured using the Lustre file system.

Characteristics:
- Distributed file system
- MDS distributes and manages the metadata, while OSS distributes and manages the data entity
- The data entity can be striped to more than one OSS

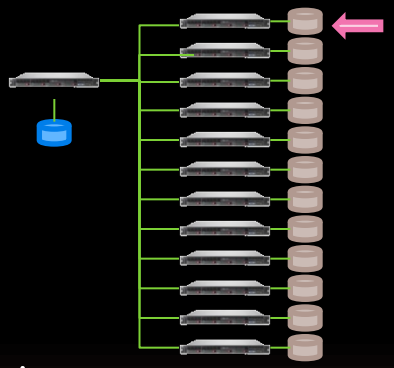
- High-speed access to few files of a few gigabytes
- Operation of several files is slower than that of ext4



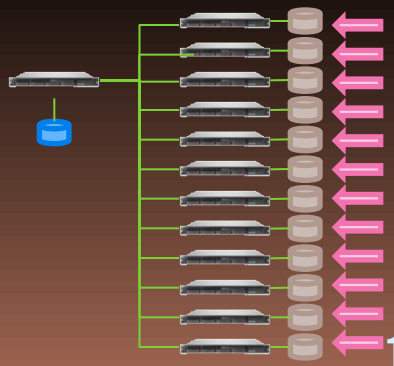
Changing stripe count (1)

- The number of OSSs in which files are allocated can be changed (the number of object storage targets is changed).
- If the stripe count is 1 (default), the content of the file exists on one OSS.
- If the stripe count is 2 or more, the content of the file is distributed on the specified number of OSSs.

When the stripe count is 1



When the stripe count is 12



Changing stripe count (2)

- Stripe count is changed by executing the lfs command

`lfs setstripe -c` number of stripes target directory
`-c:` The number of stripes can be specified freely.

```
$ mkdir stripe1
$ lfs getstripe stripe1
stripe1
stripe_count:   1 stripe_size:    1048576 stripe_offset:  -1
$ mkdir stripe12
$ lfs setstripe -c 12 stripe12
$ lfs getstripe stripe12
stripe12
stripe_count:   12 stripe_size:    0 stripe_offset:  -1
```

- By default, the stripe count is 1. Accordingly, data can be accessed at approximately 400 MB/s.
- If the stripe count is set to 12, the data can be accessed at approximately 1 GB/s.

Standard for changing stripe count

- If the change is valid
  - When a file of a few gigabytes or more is allocated
  - When a few hundred files of a few megabytes are allocated
    If a few thousand files are allocated, it is necessary to distribute them depending on the directory
    >> Data access is speeded up almost doubly, and the load of OSS is reduced
- If the change is invalid
  - When a few thousand files of few bytes to kilobytes are allocated
    >> Data access is delayed almost doubly, and the load of MDS and OSS is increased

Handling a few thousand files

- Lustre is not specialized for handling a large number of files, particularly those that are created by executing ″ls -l″

Empty files are continuously created by the touch command in the loop.
After the files are created, “ls -l” is executed.

Number of files	Creation time (s)	“ls -l” execution time (s)
1,000	1.2	0.2
10,000	12.8	1.9
100,000	148.6	19.3
* For HDD (ext4)		
100,000	118.8	0.9

- If files are being accessed by a single process, no problem occurs; however, if files are accessed by several process, the I/O delay increases and MDS is overloaded.
- MDS delay due to additional files affects other users as well
  >> Set the number of files in one directory to approximately 5,000, and divide and manage the directory.

Contents

2.1 Characteristics of each calculation node

2.2 Characteristics of file system

2.3 Upper limit value when job is entered into UGE

Upper limit of number of jobs entered into UGE

- In the following cases, an error is generated when qsub is executed:

  - When the total number of jobs awaiting execution exceeds 50,000 jobs during execution of all users
  - When the total number of jobs awaiting execution exceeds 5,000 jobs during execution of one user

- Counting jobs when special jobs are entered:

  - An MPI job is counted as one job regardless of parallelism
  - An array job is counted as one job regardless of the task count
  - The upper limit of the task count of an array job is 75,000

  >> When a few thousand jobs are to be entered, ensure that they are entered as array jobs

Upper limit of simultaneous jobs (1)

- A user with an account for general research can use up to 100 job slots simultaneously (*the restriction is relaxed)
- A user with an account for large-scale usage can use up to 500 job slots simultaneously (*the restriction is relaxed)
- A job entry using qsub can be normally carried out up to the upper limit value of the entered quantity; however, the jobs are promoted while using job slots within the restriction shown above.

Upper limit of simultaneous jobs (2)

- The restriction status can be checked using the qquota command.
  If a job falls under more than one restriction, a stricter restriction is prioritized.
  In the example shown below, 100 is applied as the upper limit.

```
$ qquota
resource quota rule limit                filter
--------------------------------------------------------------------------------
general_limit/1    slots=2/100           users lect01
large_limit/1      slots=2/500           users lect01
$
```

Contents

1. Before entering multiple jobs
2. Configuration/characteristics of super computer system
3. Finding the load property of job to be entered
4. Finding the method for managing the execution statuses of jobs

Contents

3.1 Measuring the load of a single job

3.2 Forecasting the load of job group

3.3 Reducing I/O load

3.4 Considering qsub option

3.5 Request for executing a job

Measuring the load of a single job (1)

- For measuring the load of a job, the job is interactively executed with a node of qlogin destination, or by entering a job in debug.q.
- CPU

  - Whether there is an operation of multi-thread
  - Whether multiple processes are started
    >> Checked using top command

- Memory

  - Maximum virtual memory size to be used is measured
    >> After a job is entered in debug.q, it is checked using the qreport command

- I/O

  - I/O amount (B/s) and number of file operations are checked
    >> Checked by using wrapper

Measuring the load of a single job (2)

- Measurement of CPU
  While a job is executed in the background at the qlogin destination, it is checked using the top command.

```
top -u uid -H
  -u uid:  only the process of the specified user ID is displayed
  -H:  each thread is separately displayed
```

```
$ ./job01.sh &
$ top -u lect01 -H
```
(If there are multiple commands that are executed in `job01.sh`, they are treated as multi-threads or multi-processes)

●Check the job that is being executed for the first time carefully.

Measuring the load of a single job (3)

- Measurement of memory:
  Enter a small-scale job in debug.q.
  Record the job ID at that time.

```
$ qsub -l debug job01.sh
Your job 1905176 ("job01.sh") has been submitted
```

- After the job is terminated, check it using qreport command (there are up to 5 minutes until the job can be checked using qreport after it is terminated)

```
$ qreport -j 1905176|grep maxvmem
maxvmem                2.1G
```

- The load of job should be measured using the least amount of data
- If this is done, then perform the measurement more than once while increasing the amount of data
- Check the mutual relationship between the amount of data and the consumed memory, and use it as a reference for forecasting consumed memory for actual data to be processed

Measuring the load of a single job (4)

- Measurement of I/O
  Perform the measurement using wrapper

`lustre_analyzer.rb "command arg1 arg2 …"`

- Estimate the number of file operations and I/O amount generated in `lustre` until the command specified by "`command`" is terminated. The content of measurement is shown in the table below.
- If another process accesses `lustre` at the same time in the same host, the sum can be obtained.
- The measurement result is output to the standard error output.

open	Number of opened files
close	Number of closed files
getattr	Number of obtained file attributes
setattr	Number of set file attributes
write_bytes	Amount of written data (B)
read_bytes	Amount of read data (B)

Measuring the load of a single job (5)

- Example of execution

```
$ lustre_analyzer.rb "ls -l /usr/local/seq/blast/ncbi/" > /dev/null
-------------The number of Lustre file operations-------------
    fs    open   close getattr setattr write_bytes read_bytes
lustre1     1.0     1.0  2241.0     0.0   726.0 67169280.0
lustre2     0.0     0.0     1.0     0.0     0.0     0.0
-------------------------------------------------------------
INFO:   Execute time is 0.4 seconds.
NOTICE: This infomation includes whole operation in this host
-------------------------------------------------------------
$
```

Measuring the load of a single job (6)

- Precaution when executing wrapper

- The overview of wrapper operation is as follows:
  1. Before a command given by the argument is executed, information related to lustre under /proc is obtained.
  2. The command given by the argument is executed.
  3. The information related to lustre under /proc is obtained again and the difference with the information obtained in step 1 is output.

- As the operation is carried out as shown above, the measurement may be affected by other users at the node at the qlogin destination. Moreover, it is more likely for a time-consuming job to be affected by other users.
- A method of measurement by entering a job in debug.q is as shown below.

```
$ qsub -cwd -l debug -S /usr/local/bin/ruby (other necessary option)
/usr/local/bin/lustre_analyzer.rb "./job01.sh"
```

Contents

3.1 Measuring the load of a single job

3.2 Forecasting the load of job group

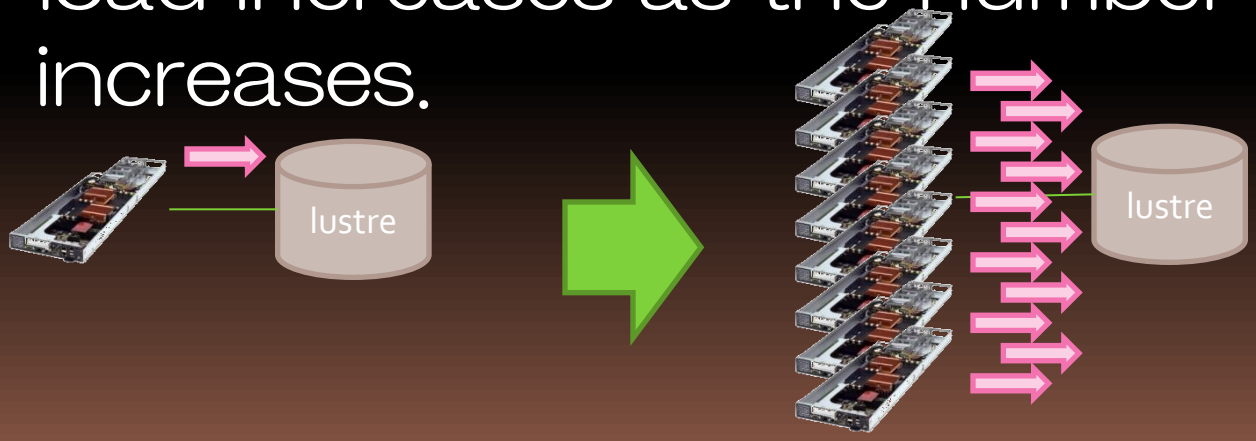
3.3 Reducing I/O load

3.4 Considering qsub option

3.5 Request for executing a job

Forecasting the load of job group (1)

- For I/O load, the value measured for a single job test needs to be multiplied by those of the simultaneous jobs.
- The load for one job is small, but the load increases as the number of jobs increases.



Forecasting the load of job group (2)

- This is estimated on the basis of the execution example of lustre_analyzer.rb described earlier.

Type	Single body measurement value	Simultaneous jobs 10	Simultaneous jobs 100	Simultaneous jobs 1000
open	1	10	100	1000
close	1	10	100	1000
getattr	2,241	22,410	224,100	2,241,000
setattr	0	0	0	0
write_bytes	726 B	7 KB	70 KB	700 KB
read_bytes	64 MB	640 MB	6.4 GB	64 GB

Allowable value of I/O load

- The allowable value of I/O load for each file system is shown below:

  - open
    Total open count per second
    Dangerous at 30,000
  - close
    Total close count per second
    Dangerous at 30,000
  - getattr
    Total getattr count per second
    Dangerous at 8,000
  - setattr
    Total setattr count per second
    Dangerous at 1,600

  - Note that this value is the
    "allowable value of load for the entire file system."
    The value of "you" + "all others" should fall within the values shown above.

Contents

3.1 Measuring the load of a single job

3.2 Forecasting the load of job group

3.3 Reducing I/O load

3.4 Considering qsub option

3.5 Request for executing a job

Reducing I/O load (1)

- If the amount of I/O is large, consider correcting the corresponding algorithm first.
- The measurement result of processing when the same operation is mounted in different methods is shown below.

* Example of mounting 1 (good)

```
$ ./lustre_analyzer.rb "./sample_good.pl"
-------------The number of Lustre file operations-------------
    fs    open   close getattr setattr write_bytes read_bytes
lustre1      0.0      0.0      8.0      0.0    363.0 24312832.0
lustre2      4.0      4.0      3.0      0.0 2450000.0 4178048.0
--------------------------------------------------------------
INFO:   Execute time is 0.1 seconds.
NOTICE: This infomation includes whole operation in this host
--------------------------------------------------------------
```

open	4
close	4
getattr	3
setattr	0
write_bytes	2.4MB
read_bytes	4.0MB

* Example of mounting 2 (bad)

```
$ ./lustre_analyzer.rb "./sample_bad.pl"
-------------The number of Lustre file operations-------------
    fs    open   close getattr setattr write_bytes read_bytes
lustre1      1.0      1.0      9.0      0.0 124726.0 8288446464.0
lustre2 250003.0 250003.0 250002.0      0.0 2450000.0 4178048.0
--------------------------------------------------------------
INFO:   Execute time is 44.3 seconds.
NOTICE: This infomation includes whole operation in this host
--------------------------------------------------------------
```

open	250,003
close	250,003
getattr	250,002
setattr	0
write_bytes	2.4MB
read_bytes	4.0MB

Reducing I/O load (2)

- The only difference in the examples of mounting 1 and 2 is that “open” exists in or out of loop.
- “open” and “close” of a file are processed with a comparatively high cost, and hence the difference is increased.

＊Example of mounting 1 (good)

```
sub do_good{
  open(INPUT,"<$IN");        # File handle for input is created
  open(OUTPUT,">>$OUT");     # File handle for output is created
  while($line=<INPUT>){      # Processing is carried out for one line
at once
    if($line =~ /foo/){      # Data is written
      print OUTPUT $line;    # if foo is contained
    }
  }
  close(OUTPUT);             # File handle is closed
  close(INPUT);              # File handle is closed
}
```

＊Example of mounting 2 (bad)

```
sub do_bad{
  open(INPUT,"<$IN");        # File handle for input is created
  while($line=<INPUT>){      # Processing is carried out for one line at
once
    if($line =~ /foo/){      # If foo is contained,
      open(OUTPUT,">>$OUT"); # file handle for output is created
      print OUTPUT $line;    # Data is written
      close (OUTPUT);        # File handle is closed
    }
  }
  close(INPUT);              # File handle is closed
}
```

- Ensure that “processing with high cost” is not executed in the loop.
- Remove “processing with high cost” that does not need to be executed in or out of the loop.

Reducing I/O load (3)

- Consider the utilization of SSD.
  For the execution host that can be used when “-l ssd” is specified, SSD is mounted on the directory /ssd.
- Access is enabled at a higher speed than with /tmp, /lustre1, or 2. The capacity is approximately 350 GB (effective capacity).
- It is the local area on the calculation node, which can be referenced only from the host for which a job is executed.
- The effect can be expected in the following cases:

  - When several intermediate files are created
  - When small-size input files are large in number

Reducing I/O load (4)

- Predefined significance when SSD is used:

  - Create the directory /ssd/user ID/ and allocate files in it (e.g., /ssd/lect01/foo)
  - The use amount is not restricted; however, control the use up to approximately 100 GB, considering simultaneous use by other users.
  - Ensure that the process of deleting files you created is added to the process of job termination
  - The directory is used for allocating files temporarily, so permanence is not guaranteed. If the server is rebooted for maintenance or some other purpose, the files are deleted (* in review)
  - If a constant time (month series queue: 1 month; week series queue: 2 weeks) passes after the final access time, the files are deleted (* in review)

Restricting jobs that are simultaneously executed (1)

- If the I/O amount is still large, suppress the simultaneous jobs.
  This method is most effective for suppressing the total I/O amount.

* For an array job

```
qsub -tc simultaneous jobs -t 1-X:Y job
  -tc simultaneous jobs: specify the upper limit of simultaneous jobs for array job
  -t: specify the usual array job
```

For the specification shown below, 500 tasks operate; however, the simultaneously operated tasks are up to 100.

```
$ qsub -tc 100 -t 1-1000:2 job01.sh
```

Restricting jobs that are simultaneously executed (2)

* For a non-array job

- Suppressing the count for executing qsub at once
- Setting up an order for jobs

`qsub -N` job name `-hold_jid` job name that has been entered `job02.sh`

`-N` job name:

Specify the name of a job that is entered them

`-hold_jid` job ID or job name that has been entered:

Until the job of the specified job `ID` or job name is terminated, the job that is entered then is `held`. If the specified job is terminated, the holding status is automatically released and the job can be executed.

In the following case, jobs are executed in order of job1, job2, job3,…

```
$ qsub -N job1 job01.sh
$ qsub -N job2 -hold_jid job1 job02.sh
$ qsub -N job3 -hold_jid job2 job03.sh
```

Contents

3.1 Measuring the load of a single job

3.2 Forecasting the load of job group

3.3 Reducing I/O load

3.4 Considering qsub option

3.5 Request for executing a job

Considering qsub option related to CPU/memory

- To investigate the amount of CPU and memory usage, the execution plan is created by directly using the measurement result of a single job.

* CPU

Does it operate for multi-threads?

Yes	″-pe def_slot thread count″ is given
No	There is no option that is specially added

* Memory

What is the size of the maximum virtual memory?

Less than 4 GB	Specification of s_vmem, mem_req is changed to the required amount
4 GB	There is no option that is specially added
4 GB or more	Specification of s_vmem, mem_req is changed to required amount
64 GB to 2 TB	Specification of s_vmem, mem_req is changed to the required amount, and medium node is used
2 TB or more	Specification of s_vmem, mem_req is changed to the required amount, and fat node is used

Resource demand when def_slot is used

- If the number of slots used for def_slot is specified, the resource requirement increases for that portion.

def_slot	s_vmem, mem_req	Actual resource requirement
None	None (default 4G is applied)	4 GB
-pe def_slot 2	None (default 4G is applied)	8 GB
-pe def_slot 4	None (default 4G is applied)	16 GB
None	-l s_vmem=8G, mem_req=8G	8 GB
-pe def_slot 4	-l s_vmem=8G, mem_req=8G	32 GB

Ensure that the actual resource requirement does not exceed the memory amount of the node to be used. If it exceeds the memory amount of the node, the job is not executed when it is submitted.

As GPU is one piece per GPU node, do not specify def_slot and “-l gpu” as a combination.

Contents

3.1 Measuring the load of a single job

3.2 Forecasting the load of job group

3.3 Reducing I/O load

3.4 Considering qsub option

3.5 Request for executing a job

Request for executing a job

Do not execute a lot of jobs suddenly

Carry out load measurement and small-scale tests using debug.q and check whether the system operates problem-free.
After this, enter jobs in the actual queue and increase the jobs to be entered gradually.

Contents

1. Before entering a lot of jobs
2. Configuration/characteristics of super computer system
3. Finding the load property of job to be entered
4. Finding the method for managing the execution statuses of jobs

Contents

4.1 Checking the execution status of job

4.2 Managing the execution result of job

Checking the execution status (1)

- Check the execution status of a job using the qstat command
- Check the load status of a calculation node using the qhost command (details can be checked using “qstat -f”)

```
qhost -u user ID
  -u: job of the specified user is displayed
```

```
$ qhost -u lect01
HOSTNAME                ARCH         NCPU NSOC NCOR NTHR  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------------------
global                  -               -    -    -    -     -       -       -       -       -
fi00                    lx-amd64      736   92  736  736  0.02 9590.6G   72.9G    8.0G     0.0
m1i                     lx-amd64       80    8   80   80  1.00 2019.8G  333.4G   20.0G     0.0
(略)
t292i                   lx-amd64       16    2   16   16  0.45   62.9G    3.6G    8.0G   43.3M
 job-ID  prior   name       user         state submit/start at     queue      master ja-task-ID
   ------------------------------------------------------------------------------------------
2160251 0.50000 job01      lect01         r     05/02/2012 10:46:57 week_hdd.q MASTER
t293i                   lx-amd64       16    2   16   16  0.48   62.9G    2.3G    8.0G   40.5M
(略)
```

＊ Note LOAD (load average) and SWAPUS (swap use amount)

Checking the execution status (2)

- If LOAD exceeds NCPU (number of CPU cores)
  >> A job may operate for multi-thread unintentionally
  >> the machinefile specification for an MPI job may fail
- If SWAPUS is set by a GB unit
  >> the s_vmem and mem_req specifications may fail

Checking the execution status (3)

- The load of the supercomputer can also be checked from the DDBJ Web page, "supercomputer operating status"

  http://www.ddbj.nig.ac.jp/system/supercom/supercom-util.html

Contents

4.1 Checking the execution status of job

4.2 Managing the execution result of job

Managing the execution result of job (1)

- If many jobs are executed, it takes time to check whether all the jobs are completed normally.
- The execution results can be collectively checked using the qreport command.

```
qreport -o user ID -N job name -b YYYYMMDDhhmm -l
  -o: only the job of the specified user ID is displayed
  -N: only the job of completely matched job name is displayed
  -l: displayed in list display mode
  -b: only the job after the specified date is displayed
  -e: Only the job before the specified date is displayed. The method for
      specifying is same as that for "-b."
```

- By using qreport command

  - The list of jobs in which a problem is generated during execution can be easily created
  - The recommended option at re-enter can be obtained (* There is no function for re-entering a job)

Managing the execution result of job (2)

- The main items of qreport output result are shown below:

task	Task ID Of MPI job, array job
ext	Exit status code of job. Number other than 0 indicates abnormal termination.
fail	Termination code of UGE. Number other than 0 indicates abnormal termination.
clock	Execution time (required time from start to termination of job)
mmem	Maximum value of actually used virtual memory
Rq, Rm	Queue when a job is re-entered (displayed only when the job results in failure)
Rm	Required memory amount that is recommended when a job is re-entered (displayed only when the job results in failure)
Ropt	qsub option that is recommended when a job is re-entered (displayed only when the job results in failure)

Contact information

- If you have any enquiries or opinions, please contact

Supercomputer SE team, National Institute of Genetics

Mail: sc-info@nig.ac.jp

Room: w202

Extension: 9461

http://www.ddbj.nig.ac.jp/system/supercom/supercom-intro.html

Revision history

Date of revision	Revised content
May 10, 2012	Newly created
Feb 22, 2013	Corrected “stripe size” to “stripe count”