Local queues without agonizing pain (maybe ...)

C. Bonati

Dipartimento di Fisica e Astronomia & INFN, Firenze

Pisa 24/11/2016

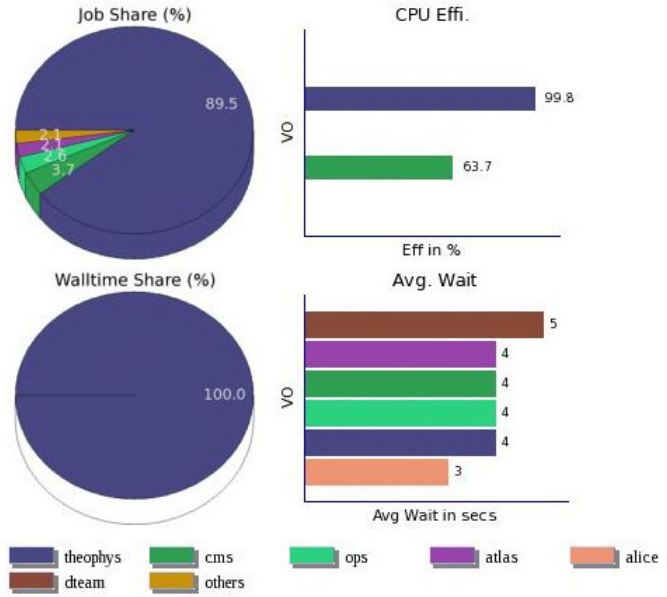
Grid for experiment, local queues for theory?

LSF Monitoring at Pisa

Current Status | **Last 6 Hours** | Last 12 Hours | Last Day | Last Week | Last Month | Last 3 Months | Last 6 Months | Last Year | Full Period

Jobs completed during the last 6 hours

VO/Group	Total Jobs	Succ Jobs	Succ Rate(%)	Walltime (sec)	CPU Time (sec)	CPU Eff(%)	Walltime Share(%)	Avg Wait (sec)
alice	12	12	100.00	1197	195	16.29	0.00	3
atlas	14	14	100.00	3867	220	5.71	0.01	4
cms	24	24	100.00	2200	1402	63.75	0.01	4
dteam	2	2	100.00	40	5	12.61	0.00	5
ops	17	17	100.00	1464	305	20.88	0.00	4
theophys	586	586	100.00	33864787	33800353	99.81	99.97	4



Last updated on 2008-09-29 08:09:59 — Developed by: S. Taneja, S. Sarkar - INFN-Pisa

The basic commands

In the following I will restrict to the three basic commands needed to use the local queues

- bsub to submit jobs
- bjobs to check jobs
- bkill to kill jobs

the idea is to explain how to use them to perform actual computations, without entering into the details of the "computer science" that is hidden behind them.

Starting point: login to `localui*.pi.infn.it` ($* = 1, 2, 3$) or `gridui3.pi.infn.it`

```
ssh -X bonati@gridui3.pi.infn.it
```

The simplest of the local queues: the interactive

"fai" stands for "farm di analisi interattiva" = "interactive analysis farm".

If you have to do something that requires more than 5min and/or more than 100MB of RAM, do NOT use the front-end, submit an interactive job:

```
bsub -Is -q fai /bin/bash -l
```

This gives you access to an interactive environment: everything works as on you PC but you are using a computing core of the cluster.

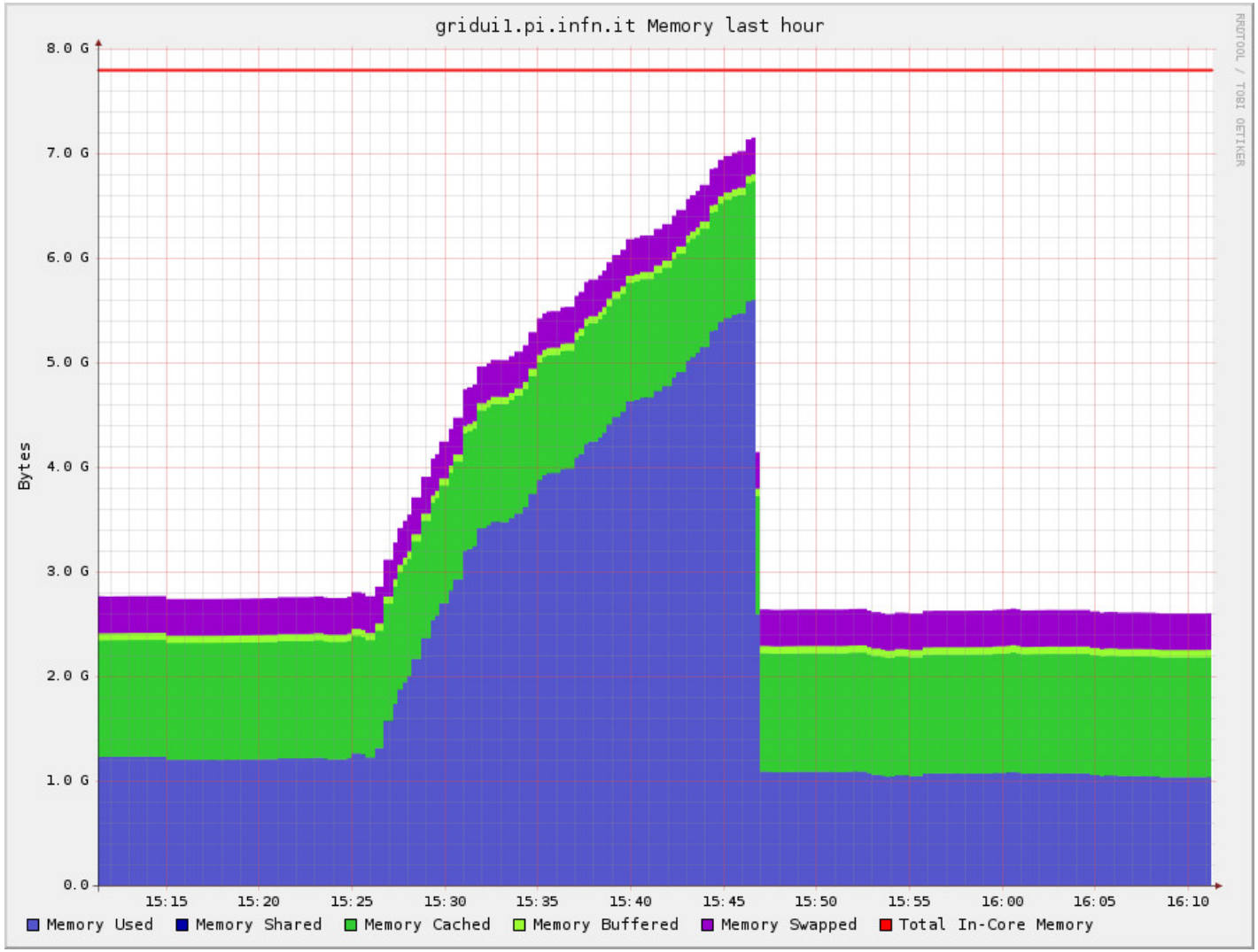
To terminate the job just run `exit`

If you need, e.g., 4 cores (1core $\leftrightarrow$ 1GB of RAM)

```
bsub -Is -n 4 -q fai /bin/bash -l
```

Easy and fast, no reason not to use the interactive farm
(only if you hate the other users).

Explicit example of front-end misuse



"Thanks" to C.V.

Local queue (no I/O from/to file)

The test code:

```
#include<stdio.h>
#include<stdlib.h>

int main (int argc, char **argv)
    {
    printf("Hello world!\n");
    return 0;
    }
```

Compile code:

```
cd /home/users/bonati/test_vari/hello
gcc -O3 hello.c -o hello
```

Submit job (wrong):

```
cd /home/users/bonati/test_vari/hello
bsub -q local -o out.txt -e err.txt -J test hello
```

Local queue (no I/O from/to file)

Syntax:

```
bsub -q queue_name [-o stdout_file] [-e stderr_file] \\
     [-J identifier] executable
```

the terms in square brackets are optional.
Check that the job is waiting/running in the queue

```
-bash-4.1$ bsub -q local -o out.txt -e err.txt -J test hello
Job <9467038> is submitted to queue <local>.
-bash-4.1$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
9467038    bonati  PEND  local      gridui3                 test       Nov 16 13:54
-bash-4.1$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
9467038    bonati  RUN   local      gridui3     se1wn24     test       Nov 16 13:54
-bash-4.1$ bjobs
No unfinished job found
```

If you realize some job is not working properly and you want to kill it, use

```
bkill JOBID_value
```

Local queue (no I/O from/to file)

Something went wrong:

```
-bash-4.1$ cat err.txt
./.lunch_9467038.sh: line 1: hello: command not found
```

Reason of the problem: where is in the filesystem the executable "hello"?

Submit job (OK):

```
cd /home/users/bonati/test_vari/hello
bsub -q local -o out.txt -e err.txt ${PWD}/hello
```

At the end we now have

```
-bash-4.1$ cat err.txt
-bash-4.1$
```

i.e. "err.txt" is empty.

Local queue (no I/O from/to file)

```
-bash-4.1$ cat out.txt
Sender: LSF System <lsfadmin@se1wn54>
Subject: Job 9467430: <test> Done

Job <test> was submitted from host <gridui3> by user <bonati> in cluster <infn-pisa>.
Job was executed on host(s) <se1wn54>, in queue <local>, as user <bonati> in cluster <infn-pisa>.
</home/users/bonati> was used as the home directory.
</home/users/bonati/test_vari/hello> was used as the working directory.
Started at Wed Nov 16 14:05:04 2016
Results reported at Wed Nov 16 14:05:05 2016

Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
/home/users/bonati/test_vari/hello/hello
------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time   :      1.25 sec.
    Max Memory :        55 MB
    Max Swap   :        92 MB

    Max Processes  :         1
    Max Threads    :         1

The output (if any) follows:

Hello world!


PS:

Read file <err.txt> for stderr output of this job.

-bash-4.1$
```

The "input" case

A test code that requires input:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX_STRING_LENGTH 100
int main (int argc, char **argv)
    {
    char file_n[MAX_STRING_LENGTH];
    FILE *file_p;
    if(argc != 2)
      {
      printf("Usage: %s input_file\n", argv[0]);
      return 1;
      }
    if(strlen(argv[1]) >= MAX_STRING_LENGTH)
      {
      fprintf(stderr, "filename too long, increase MAX_STRING_LENGTH\n");
      return 1;
      }
    strcpy(file_n,argv[1]);
    file_p=fopen(file_n, "r");
    if(file_p==NULL)
      {
      fprintf(stderr, "Error in opening the file %s\n", file_n);
      return 1;
      }
    printf("Hello world from file '%s'!\n", file_n);
    fclose(file_p);
    return 0;
    }
```

The "input" case

Compile code:

```
cd /home/users/bonati/test_vari/hellowithinput
gcc -O3 hellowithinput.c -o hellowithinput
```

Submit job (wrong):

```
cd /home/users/bonati/test_vari/hellowithinput
touch input.in
bsub -q local -o out.txt -e err.txt \\
     ${PWD}/hellowithinput input.in
```

Same problem as before:

```
-bash-4.1$ cat err.txt
Error in opening the file input.in
```

Submit job (OK):

```
cd /home/users/bonati/test_vari/hellowithinput
touch input.in
bsub -q local -o out.txt -e err.txt \\
     ${PWD}/hellowithinput ${PWD}/input.in
```

The "output" case

A test code that produce output files:

```
#include<stdio.h>
#include<stdlib.h>

int main (int argc, char **argv)
    {
    FILE *file_p;
    file_p=fopen("output.dat", "w");
    if(file_p==NULL)
      {
      fprintf(stderr, "Problem in opening output file\n");
      return 1;
      }
    fprintf(file_p, "Hello world!\n");
    fclose(file_p);
    return 0;
    }
```

The "output" case

Compile code:

```
cd /home/users/bonati/test_vari/hellowithoutput
gcc -O3 hellowithoutput.c -o hellowithoutput
```

Submit job (wrong):

```
cd /home/users/bonati/test_vari/hellowithoutput
bsub -q local -o out.txt -e err.txt \\
     ${PWD}/hellowithoutput
```

`err.txt` is empty but `output.dat` was not created in the directory.

In fact it was created in `/tmp/bonati/LSF_9468590`

One Script to bring them all

The solution of all the "locality" problems: make a script!

Example of script:

```
#!/bin/bash

cd /home/users/bonati/test_vari/hellowithinput
touch input.in
./hellowithinput input.in

cd /home/users/bonati/test_vari/hellowithoutput
./hellowithoutput
```

Submit the script:

```
chmod +x script.sh
bsub -q local -o out.txt -e err.txt ${PWD}/script.sh
```

Do I need a specific version of the compiler?

Sometimes the last version of a compiler is needed, e.g. because

- more aggressive optimization strategies are applied
- better checks of the code are performed to signal possible flaws
- recent language extension are implemented

On gridui3 `gcc version 4.4.7` is installed

For more recent versions:

```
. /afs/cern.ch/sw/lcg/external/gcc/4.9/ \\
            x86_64-slc6-gcc49-opt/setup.sh # for GCC 4.9

. /afs/cern.ch/sw/lcg/external/gcc/6.2/ \\
            x86_64-slc6-gcc62-opt/setup.sh # for GCC 6.2
```

Queues details

fai	walltime limit: 24h
local	walltime limit: 48h
parallel	walltime limit: 6h. Max. number of jobslot available: 8
longparallel	walltime limit: 24h. Max. number of jobslot available: 16
compilation	walltime limit: 1h

For parallel and longparallel
1 jobslot = 16 cores
32 cores per node

To use parallel/longparallel queues

```
bsub < script.sh
```

with a script that is something like

```
#!/bin/bash

#BSUB -J jobname
#BSUB -n jobslot_number
#BSUB -q longparallel
#BSUB -G iniziativa_specifica_infn
#BSUB -o jobout_%J
#BSUB -e joberr_%J

cd directory
mpirun executable
```

If you want 2 jobslots on the same node add

```
#BSUB -R "span[hosts=1]"
```

If jobslot%2 == 0 an you want whole nodes use

```
#BSUB -R "span[ptile=2]"
```

Enjoy your runs!