PEC 1

Pablo Riutort Grande

April 16, 2023

M0.538 - HIGH PERFORMANCE COMPUTING

MU Ingeniería Informática / MU Ingeniería Computacional y Matemática

Estudios de Informática, Multimedia y Telecomunicación

Contents

1 Performance evaluation: Sample parametric study

1.1 Provide the SGE script(s) and the shell scripts (or methodology, e.g., program, command sequence) that you used to carry out the parametric study's executions systematically. Describe your choices (e.g., number of jobs launched vs. combinations in each job).

To do this exercise, 5 SGE scripts were developed, one for each call to app with different input parameters (10, 100, 500, 1000, 1500).

Figure 1: Contents of HPC's home/PEC1 folder

```
[hpc141@eimtarqso PEC1]$ ls -1
1000.sge
100.sge
10.sge
1500.sge
500.sge
app
app.c
lib.o
README.md
results_1
results_10
results_2
results_3
results_4
results_5
results_6
results_7
results_8
results_9
run.sh
sge.err
[hpc141@eimtarqso PEC1]$ _
```

Each SGE script contains the following parameters:

- **#$ -cwd**: Directs SGE to run the job in the same directory from which you submitted it.
- **#$ -S /bin/bash**: Specifies the interpreting shell for the job.
- **#$ -N job_name**: Sets the name of the job, in our case is specified with leading underscore

followed by the input parameter (_<number>).

- **#$ -o /dev/null**: Redirect standard output to null in order to avoid printing.
- **#$ -e sge.err**: Redirect error output to sge.err file

At the end of the SGE script we will find the command to run, each command will call the time command (not bash's) followed by a call to app with a given number. The time command will have the next parameters:

- **-f %e**: According to "time" manual, %e will give "Elapsed real time (in seconds)"
- **-o results/<number>.txt**: Specifies the output file. One file per each input parameter in app.

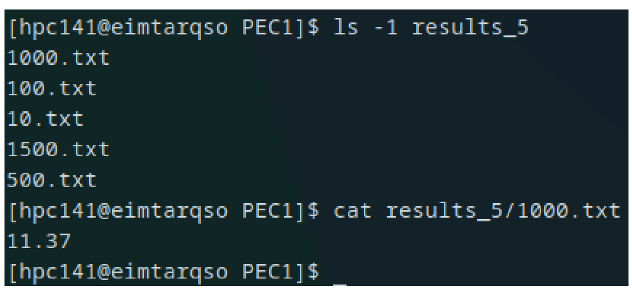
```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _10
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/10.txt ./app 10
```

Listing 1: Contents of 10.sge script

A run script helper was developed that calls each SGE script. For every call to this script a renaming of the directory results is needed in order to not let the next execution overwrite results, hence, we have results_1, results_2, and so on for every execution of the run script, 10 in total.

Figure 2: Contents of results in fifth execution and 1000.txt file



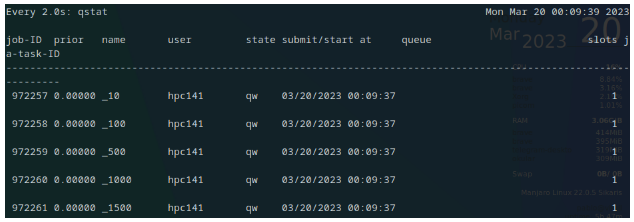

```
#!/bin/sh

mkdir results
for j in 10 100 500 1000 1500
do
  qsub "$j.sge"
done
watch -n 2 -d qstat
```

Listing 2: Contents of run.sh script

run.sh will iterate over the numbers 10, 100, 500, 1000 and 1500 and run each corresponding SGE script. Then will call the watch command to check the qstat command every 2 seconds to get a view of the jobs execution.

Figure 3: Execution of last line of run.sh script: watch command



1.2 Provide a plot of the execution time of the application with the requested input parameters. It is requested to include the average and standard deviation (or the percentiles/quartiles of your choice) of various (e.g., 5-10) executions.

At this point, a Python script was made to treat the results and to plot a box diagram. This script has two dependencies:

- Pandas: a software library written for the Python programming language for data manipulation and analysis
- matplotlib: a comprehensive library for creating static, animated, and interactive visualizations in Python

```python
import pandas as pd

from matplotlib import pyplot as plt
from os import walk

data = {}
# get data from results files
for dirpath, dirnames, filenames in walk("."):
    for filename in filenames:
        if filename.endswith(".txt"):
            key = int(filename.split(".")[0])
            with open(filename, "r") as _file:
                data[key] = [float(i) for i in _file.readlines()]

df = pd.DataFrame(data)
df = df.reindex(sorted(data.keys()), axis=1)

p = df.plot.box()
p.set_title("Sample parametric study")
p.set_xlabel("app parameter")
p.set_ylabel("Time (seconds)")
plt.show()
```

Listing 3: Python program to treat data results

**For this exercise all results were placed in common files at same location of next script.*

This script loads all data from given txt files and build a Pandas DataFrame object. After this point, we need to reoder the indexes to have them in ascending order, label axis and give it a title to get our plot.

Below some tables show what information Pandas was able to retrieve with the provided data from result files:

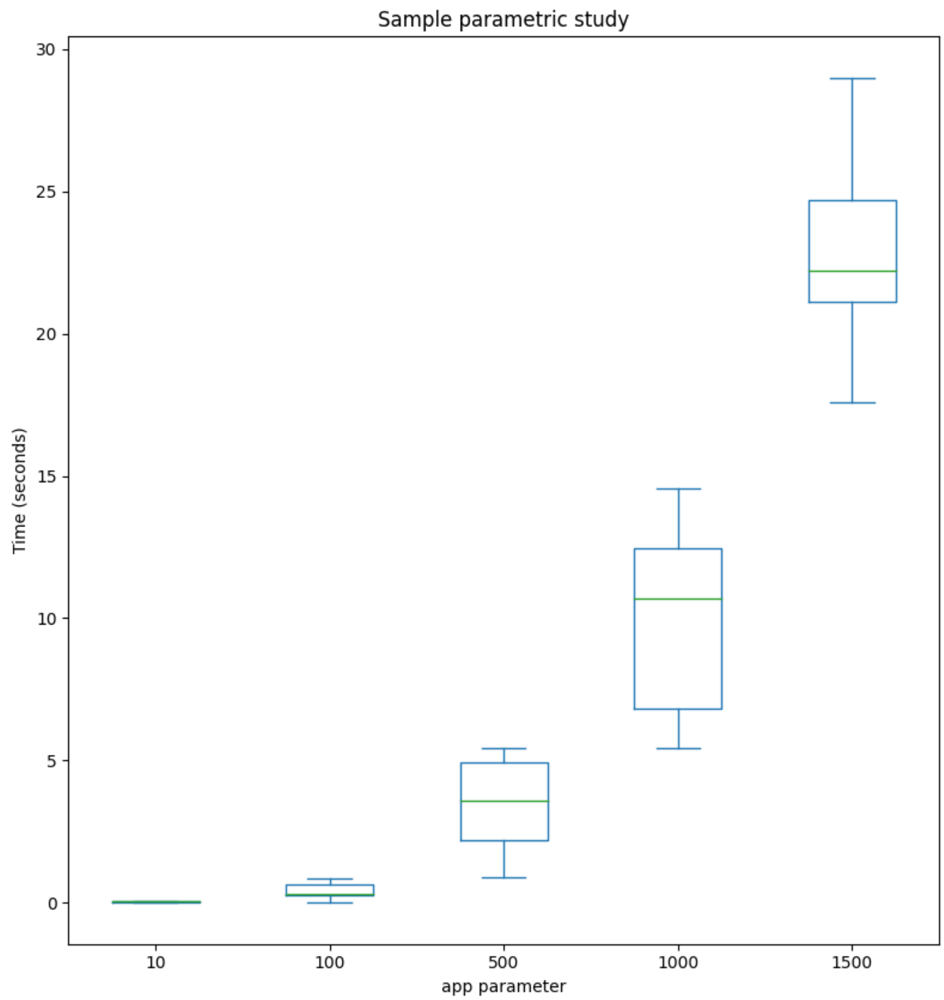
Execution Input	**10**	**100**	**500**	**1000**	**1500**
1	0.05	0.85	3.26	14.57	22.15
2	0.00	0.30	1.45	13.48	29.00
3	0.06	0.66	3.30	11.55	22.28
4	0.01	0.01	0.90	12.76	17.60
5	0.01	0.31	4.49	9.99	20.89
6	0.05	0.25	5.42	11.37	24.88
7	0.04	0.74	3.86	8.01	28.62
8	0.02	0.22	5.27	6.27	24.13
9	0.05	0.65	5.08	5.45	20.51
10	0.04	0.24	1.82	6.42	21.79

Table 1: Values of time command for each execution and input parameter.

	10	**100**	**500**	**1000**	**1500**
count	10	10	10	10	10
mean	0.033000	0.423000	3.485000	9.98700	23.185000
std	0.021108	0.277611	1.643359	3.27079	3.567923
min	0.000000	0.010000	0.900000	5.45000	17.600000
25%	0.012500	0.242500	2.180000	6.81750	21.115000
50%	0.040000	0.305000	3.580000	10.68000	22.215000
75%	0.050000	0.657500	4.932500	12.45750	24.692500
max	0.060000	0.850000	5.420000	14.57000	29.000000

Table 2: Extracted data from Table 1.

Figure 4: Execution time of the application with the requested input parameters



2 Queuing System and Scheduling

2.1 FCFS

In FCFS scheduling, the process that arrives first gets executed first, and so on. The CPU is allocated to the first process in the ready queue, and it continues to execute that process until it is either completed or blocked.

$$\text{Utilization} = \text{Used slots / Total slots} = \frac{255}{350} = \mathbf{72.85\%}$$

CPU\Time	1	2	3	4	5	6	7	8	9	10	11
CPU 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 2	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 3	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 4	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 5	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 5	Job 5	
CPU 6	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 7	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 8	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 9		Job 2	Job 2	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3
CPU 10		Job 2	Job 2								

Table 3: FCFS from time slot 1 to 11

CPU\Time	12	13	14	15	16	17	18	19	20	21
CPU 1	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 2	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 3	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 4	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 5		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9		
CPU 6		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9		
CPU 7		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9		
CPU 8		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9		
CPU 9		Job 6	Job 6	Job 6						
CPU 10		Job 7	Job 7	Job 7	Job 7					

Table 4: FCFS from time slot 12 to 21

CPU\Time	22	23	24	25	26	27	28	29	30
CPU 1	Job 10	Job 10	Job 11	Job 11	Job 11	Job 11			Job 16
CPU 2	Job 10	Job 10	Job 11	Job 11	Job 11	Job 11			Job 16
CPU 3	Job 10	Job 10	Job 12	Job 12	Job 12	Job 12			Job 16
CPU 4	Job 10	Job 10	Job 12	Job 12	Job 12	Job 12			Job 16
CPU 5	Job 10	Job 10	Job 13	Job 13	Job 13	Job 13			Job 16
CPU 6	Job 10	Job 10	Job 14	Job 14	Job 14	Job 14			
CPU 7	Job 10	Job 10	Job 15	Job 15	Job 15	Job 15	Job 15	Job 15	
CPU 8	Job 10	Job 10							
CPU 9	Job 10	Job 10							
CPU 10	Job 10	Job 10							

Table 5: FCFS from time slot 22 to 30

CPU\Time	31	32	33	34	35
CPU 1	Job 16	Job 17	Job 17	Job 17	Job 18
CPU 2	Job 16	Job 17	Job 17	Job 17	Job 18
CPU 3	Job 16	Job 17	Job 17	Job 17	Job 18
CPU 4	Job 16	Job 17	Job 17	Job 17	
CPU 5	Job 16	Job 17	Job 17	Job 17	
CPU 6		Job 17	Job 17	Job 17	
CPU 7		Job 17	Job 17	Job 17	
CPU 8		Job 17	Job 17	Job 17	
CPU 9		Job 17	Job 17	Job 17	
CPU 10					

Table 6: FCFS from time slot 31 to 35

2.2 EASY-backfilling

In EASY-backfilling, the scheduler first applies the FCFS policy to the incoming processes. If a new job arrives while an existing job is being executed, and if the new job can be scheduled to complete before the existing job finishes, the new job is backfilled in the remaining time of the existing job.

This approach helps to minimize the wait time of short processes by allowing them to get executed ahead of long processes that arrived earlier, and also maximizes the utilization of the CPU by filling up the remaining time of longer processes with shorter processes.

$$\text{Utilization} = \text{Used Slots} \ / \ \text{Total Slots} = \frac{252}{320} = \mathbf{78.75\%}$$

CPU\Time	1	2	3	4	5	6	7	8	9	10	11
CPU 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 2	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	**Job 4**	Job 4	Job 4
CPU 3	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 4	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 4	Job 4	Job 4
CPU 5	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 7	Job 7	Job 7
CPU 6	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 7	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 8	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1	Job 1			
CPU 9		Job 2	Job 2	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3	Job 3
CPU 10		Job 2	Job 2			Job 5	Job 5				

Table 7: EASY-backfilling from time slot 1 to 11

CPU\Time	12	13	14	15	16	17	18	19	20	21
CPU 1	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 2	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 3	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 4	Job 4	Job 6	Job 6	Job 6	Job 8	Job 8	Job 8	Job 8	Job 8	Job 8
CPU 5	Job 7	Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9	Job 16	Job 16
CPU 6		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9	Job 16	Job 16
CPU 7		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9	Job 16	Job 16
CPU 8		Job 6	Job 6	Job 6	Job 9	Job 9	Job 9	Job 9	Job 16	Job 16
CPU 9		Job 6	Job 6	Job 6					Job 16	Job 16
CPU 10										

Table 8: EASY-backfilling from time slot 12 to 21

CPU\Time	22	23	24	25	26	27
CPU 1	Job 10	Job 10	Job 11	Job 11	Job 11	Job 11
CPU 2	Job 10	Job 10	Job 11	Job 11	Job 11	Job 11
CPU 3	Job 10	Job 10	Job 12	Job 12	Job 12	Job 12
CPU 4	Job 10	Job 10	Job 12	Job 12	Job 12	Job 12
CPU 5	Job 10	Job 10	Job 13	Job 13	Job 13	Job 13
CPU 6	Job 10	Job 10	Job 14	Job 14	Job 14	Job 14
CPU 7	Job 10	Job 10	Job 15	Job 15	Job 15	Job 15
CPU 8	Job 10	Job 10	Job 18			
CPU 9	Job 10	Job 10	Job 18			
CPU 10	Job 10	Job 10	Job 18			

Table 9: EASY-backfilling from time slot 22 to 27

CPU\Time	28	29	30	31	32
CPU 1			Job 17	Job 17	Job 17
CPU 2			Job 17	Job 17	Job 17
CPU 3			Job 17	Job 17	Job 17
CPU 4			Job 17	Job 17	Job 17
CPU 5			Job 17	Job 17	Job 17
CPU 6			Job 17	Job 17	Job 17
CPU 7	Job 15	Job 15	Job 17	Job 17	Job 17
CPU 8			Job 17	Job 17	Job 17
CPU 9					
CPU 10					

Table 10: EASY-backfilling from time slot 28 to 32

A Data Files

A.1 10.txt

```
0.05
0.00
0.06
0.01
0.01
0.05
0.04
0.02
0.05
0.04
```

A.2 100.txt

```
0.85
0.30
0.66
0.01
0.31
0.25
0.74
0.22
0.65
0.24
```

A.3 500.txt

```
3.26
1.45
3.30
0.90
4.49
5.42
3.86
5.27
5.08
1.82
```

A.4 1000.txt

```
14.57
13.48
11.55
12.76
9.99
11.37
8.01
6.27
5.45
6.42
```

A.5 1500.txt

```
22.15
29.00
22.28
17.60
20.89
24.88
28.62
24.13
20.51
21.79
```

B SGE files

B.1 10.sge

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _10
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/10.txt ./app 10
```

B.2 100.sge

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _100
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/100.txt ./app 100
```

B.3 500.sge

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _500
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/500.txt ./app 500
```

B.4 1000.sge

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _1000
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/1000.txt ./app 1000
```

B.5 1500.sge

```
#!/bin/bash
#$ -cwd
#$ -S /bin/bash
#$ -N _1500
#$ -o /dev/null
#$ -e sge.err

/usr/bin/time -f %e -o results/1500.txt ./app 1500
```

C Scripts

C.1 run.sh

```
#!/bin/sh

mkdir results
for j in 10 100 500 1000 1500
do
  qsub "$j.sge"
done
watch -n 2 -d qstat
```

C.2 plot.py

```
import pandas as pd

from matplotlib import pyplot as plt
from os import walk

data = {}
# get data from results files
for dirpath, dirnames, filenames in walk("."):
    for filename in filenames:
        if filename.endswith(".txt"):
            key = int(filename.split(".")[0])
            with open(filename, "r") as _file:
                data[key] = [float(i) for i in _file.readlines()]

df = pd.DataFrame(data)
df = df.reindex(sorted(data.keys()), axis=1)

p = df.plot.box()
p.set_title("Sample parametric study")
p.set_xlabel("app parameter")
p.set_ylabel("Time (seconds)")
plt.show()
```

[Knuth(1986)]

References

[Knuth(1986)] D. E. Knuth. *The TEX Book*. Addison-Wesley Professional, 1986.