My typical workflow

Jakub Muszyński

6th–7th May 2014

Computer Science and Communications (CSC) Research Unit

My experiments

I am simulating a P2P protocol.

- Executions are **independent**.
- Each execution has a set of parameters:
  - network size — number of nodes in the network,
  - initialization — initial state of the network,
  - etc.
- Each parameter has a different set of values:
  - network size: 500, 1000, . . . nodes,
  - etc.
- **For each combination of the parameters, I need X executions.**

Implementation

- Done in Java — depends on the GraphStream[1] library.
  - Remember about the proper settings of the Java Virtual Machine.

    ↪ Especially: `-d64 -Xms$memoryNeeded -Xmx$memoryNeeded`
- State is implemented.
  - Simple implementation of the **Serializable** interface.
  - Output is compressed (**GZIP**) on the application level.

[1] http://graphstream-project.org/

Resources needed — example

- Total number of executions can be huge:
  - parameters 1 and 2 have 5 values each,
  - parameter 3 has 10 values,
  - parameter 4 has 20 values,
  - parameter 5 has 2 values,
  - for each combination of parameters, I need 100 executions.

  In total it gives: **1.000.000 independent executions**.
- Time required for a single execution:
  - **from a few minutes to a couple of hours**.
- Memory (**RAM**):
  - **up to 4 GB** (depending on the problem size).
- Input/Output operations:
  - state files,
  - final results.

Batches

- **1 batch = 1 job**
- X executions grouped by the values of the parameters.
- Created by the configuration script which:
  - creates a directory for the results (**mkdir**) of the batch: `./parameter1_value/parameter2_value/.../parameter5_value`
  - puts there the application configuration, setting appropriate parameters (**cp** and **sed**),
  - creates marker files (missing executions) (**touch**).
- Executed using **GNU Parallel**[2] — see PS2.

[2] `http://www.gnu.org/software/parallel/`

Queue

Depending on the current load of the platform:

- **default** queue (many users/jobs) with state saving:
  - before the end of the walltime if the execution is not finished.
- **besteffort** queue (few users/jobs) with state saving:
  - periodically (every X minutes)
    ↪ internally implemented in the application.
  - before the end of the walltime if the execution is not finished.

Default queue — oarsub options

- `-n $jobName`

  ↪ If you name the jobs, it is easier to manage them.

- `-t idempotent`

  ↪ Exit code equal to 99 ⇒ job is resubmitted with the same parameters.

- `-l nodes=1,walltime=$hours`

  ↪ Bash variable `hours` is set depending on the problem size:

```
problemSize=‘echo $dir | sed ’s/.*networkSize\([0-9]*\).*/\1/’‘
hours="2"
if [ $problemSize -ge 500 ]; then
    hours="4"
fi
```

- `--checkpoint 900 --signal 12`

  ↪ 15 minutes before **walltime** ends, signal 12 (**USR2**) is sent.

Besteffort queue — oarsub options

Differences:

- Add: `-t besteffort`
- Change the properties: `-l nodes=1/cpu=1,walltime=$hours`

Job submision script (simplified)

1. Find all directories with missing executions:

```
missingDirs=`find . -iname *.missing -printf "%h\n" | sort -u`
```

2. For each directory:

   - Wait for the space in the queue (do not spam with too many jobs):

```
while [ `oarstat -u jmuszynski | wc -l` -ge 32 ]; do
    echo "Waiting 10 minutes to free the queue..."
    sleep 10m
done
```

   - Setup parameters for the oarsub — like the variable `hours` previously.
   - Submit the job:

```
oarsub <all_the_parameters_described_previously>
```

1 Job = GNUParallel + checkpointing

- Trap the checkpoint signal (defined previously in the oarsub):

```
CHKPNT_SIGNAL=12
EXIT_UNFINISHED=99

function checkpointAll {
    # do not start new jobs
    kill -TERM $parallelPID
    # checkpoint running
    for p in ‘ps -fujmuszynski | grep $application\
            | grep $parallelPID | grep -v parallel\
            | awk ’{ print $2 }’‘; do
        kill -$CHKPNT_SIGNAL $p
    done
    # wait to finish, quit
    wait $parallelPID
    exit $EXIT_UNFINISHED
}

trap "checkpointAll" $CHKPNT_SIGNAL
```

GNUParallel

- Run the parallel tasks:

```
parallel -j$jobsPossible $application {} ::: $testNumbers &
parallelPID=$!
```

Besteffort jobs — WARNINGS

Besteffort jobs CAN BE KILLED AT ANY MOMENT!

- You have to accept some loss of the CPU time.
  - ↪ Walltime should be **SHORT** if you do not have the state saving.
- At **ANY** moment includes even the state saving!
  - ↪ Keep two versions of the state — previous and current.

Besteffort jobs — WARNINGS

Abount the walltime & the number of jobs

- HPC is a shared platform.

  ↪ Use a common sense when submitting the jobs.
  ↪ Limits are flexible, but avoid misuse.

Max walltime	**Max number of active jobs per user**
9000:00:00	1000

HPC $\neq$ PC

Which means, that you should monitor execution of your jobs (`https://hpc.uni.lu/status/ganglia.html`). As:

- Failures affect other users.
- Performance issues also, especially:
  - **I/O operations**,
  - **RAM** usage.

Thank you!