# LSF tips and tricks for workflow designers

## ID SIS project presentation meeting 02.03.2017

High Performance Computing Group, Scientific IT Services, ETH Zurich

## Scaling up from a single server to a cluster

- Most people develop workflows on a local server
- Porting a workflow to a cluster brings new challenges that were not visible on the local server
  - You must submit your computations as jobs to the batch system (not interactive, runtime limits)
  - You need to request the right amount of resources for your jobs (memory, scratch space, etc.)
  - You are sharing resources with other users (your bad actions may affect them)
  - Clustered file systems brings a lot of capacity but cannot compete with local storage (IOPS)
  - Even minor problems can become huge when you scale up to hundreds of jobs
- Please be very careful when scaling up (test, test and test again!)

## LSF optimization

- Overhead of jobs
  - Workstation: many short tasks can be run one after another without overhead
  - Cluster: each job has an overhead of at least 1 min (scheduling and clean up)
  - Don't run short jobs; instead, group small computations in bigger jobs
- Look at overall efficiency of the workflow, not only at the parallel part
  - Remember Amdahl's law: Amdahl's law
  - For 24 cores and a parallel part of 90%: $S(24,0.9) = \frac{1}{(1-0.9)+\frac{0.9}{24}} = 7.27$
- Run multiple jobs in parallel, instead of trying to parallelize each job
  - Many workflows are inherently parallel
  - The 24 cores from the example above could be used to run 24 jobs

## How to group small jobs

```
#!/bin/bash
for day in {1..365}; do
  for hour in {1..24}; do
    bsub command $day $hour
  done;
done;
```

```
#!/bin/bash
for day in {1..365}; do
  bsub << EOF
#!/bin/bash
for hour in {1..24}; do
  command $day \$hour
done;
EOF
done;
```

```
#!/bin/bash
bsub << EOF
#!/bin/bash
for day in {1..365}; do
  for hour in {1..24}; do
    command \$day \$hour
  done;
done;
EOF
```

ETH zürich

## How to replace loops with a job array

```
#!/bin/bash
for day in {1..365}; do
  for hour in {1..24}; do
    bsub command $day $hour
  done;
done;
```

```
#!/bin/bash
for day in {1..365}; do
  bsub -J "A[1-24]" << EOF
hour=\$LSB_JOBINDEX
command $day \$hour
EOF
done;
```

## I/O optimization

- Avoid unnecessary I/O
  - Don't write anything that does not need to be written (stdout, stderr)
  - Do not pass information via files (use other methods)
- Avoid small files whenever possible
  - Group small files in tar or zip archives
  - Use libraries like NetCDF, HDF5, XDR (buffered I/O)
- Choose the best file system for the job
  - Use local scratch for I/O intensive operations
  - Use parallel file system (/cluster/scratch, /cluster/work) for big files
- Think about I/O patterns
  - Reduce number of system calls (open, close, seek, read, write, etc.)

## Euler file systems

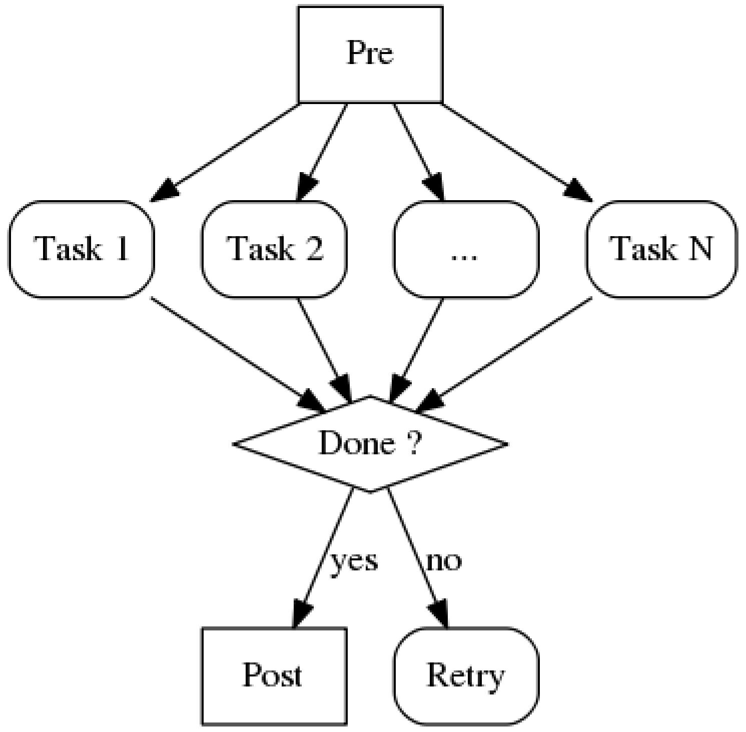
| File system | Life span | Backup | Max size | Small files | Large files |
|---|---|---|---|---|---|
| `/cluster/home` | Permanent | Yes | 16 GB | + | o |
| `/cluster/scratch` | 2 weeks | No | 2.5 TB | o | ++ |
| `/cluster/project` | 4 years | Optional | Flexible | + | + |
| `/cluster/work` | 4 years | No | Flexible | o | ++ |
| local `/scratch` | Job | No | 800 GB | ++ | o |
| central NAS | Flexible | Optional | Flexible | + | + |

ETH zürich

## Workflow management with LSF

- Do not rely on `bjobs` to monitor the progress of your jobs
  - Each call of `bjobs` queries the LSF database and create load on the batch system
  - Jobs that finished more than 1 hour ago are not visible for `bjobs`
- Do not rely on files to monitor the progress of your jobs
  - Creating and checking files creates load on the file system
  - Small files waste space (blocksize of 1 MB on /cluster/work and /cluster/scratch)
- Instead, use LSF features such as
  - Email notification to tell you when a job begins/ends (`bsub -B -N ...`)
  - Dependency conditions to define the relationship between tasks in your workflow
  - Job arrays to submit and manage similar jobs as a single entity
  - Light-weight jobs for non CPU-intensive tasks

## Dependency conditions



```
bsub -J pre ./pre

bsub -J "task[1-$N]" -w "done(pre)" ./task

bsub -w "numdone(task,*)" ./post

bsub -w "numexit(task, > 0)" ./retry
```

## Job arrays

- Multiple similar jobs can be submitted at once using a so-called “job array”
  - All jobs in an array share the same JobID
  - Use job index between brackets to distinguish between individual jobs in an array
  - LSF stores job index and array size in environment variables
  - Each job can have its own standard output
- Examples:

```
bsub -J "array_name[1-N]" ./program      # submit N jobs at once
bjobs -J array_name                      # all jobs in an array
bjobs -J jobID                           # all jobs in an array
bjobs -J array_name[index]               # specific job in an array
bjobs -J jobID[index]                    # specific job in an array
```

## Job array example

```
[leonhard@euler03 ~] bsub -J "hello[1-8]"
bsub> echo "Hello, I am job $LSB_JOBINDEX of $LSB_JOBINDEX_END"
bsub> ctrl-D
Job array.
Job <29976045> is submitted to queue <normal.4h>.
[leonhard@euler03 ~]$ bjobs
JOBID      USER      STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
29976045   leonhard  PEND  normal.4h  euler03                 hello[1]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[2]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[3]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[4]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[5]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[6]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[7]   Oct 10 11:03
29976045   leonhard  PEND  normal.4h  euler03                 hello[8]   Oct 10 11:03
[leonhard@euler03 ~]$ bjobs -J hello[6]
JOBID      USER      STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
29976045   leonhard  PEND  normal.4h  euler03                 hello[6]   Oct 10 11:03
```

## Light-weight jobs

- Light-weight jobs are jobs that do not consume a lot of CPU time, for example
  - Master process in some type of parallel jobs
  - File transfer program
  - Interactive shell
- Some compute nodes are specially configured for light-weight jobs
  - They allow multiple light-weight jobs to run on the same core at the same time
  - This is more efficient than allocating 100% of a core to a job that would use only 10%
- Use the option "-R light" to submit a light-weight job
  - Example: submit a 15-minute interactive bash shell

```
bsub -W 15 -Is -R light /bin/bash
```

  - Do not forget to logout (type "logout" or "exit") when you're done

## Light-weight job example

```
[leonhard@euler03 ~]$ bsub -W 15 -Is -R light /bin/bash
Generic job.
Job <27877012> is submitted to queue <light.5d>.
<<Waiting for dispatch ...>>
<<Starting on e2002>>
[leonhard@e2002 ~]$ pwd
/cluster/home/leonhard
[leonhard@e2002 ~]$ hostname
e2002
[leonhard@e2002 ~]$ exit
exit
[leonhard@euler03 ~]$
```

## Questions?