LSF 使用培训

Batch job & Queue

- IBM Spectrum LSF is the utility used for batch processing support, jobs must run through it.
- There are several queues present in the machines and different users may access different queues, all queues have different limits in amount of cores for the jobs and duration.
- The standard configuration and limits of the queues are the following:
- Default queue:
  - normal
  - smp

- bqueue
- bqueue -l

Queue name	prio	Maximum wallclock
normal	30	N/A
smp	30	N/A

What is IBM Platform LSF?

IBM Platform LSF is the most powerful workload manager for demanding, distributed and mission-critical high performance computing environments.

A Simple Idea:

- Virtualize a heterogeneous infrastructure
- Optimally manage workloads according to policy

Key benefits:

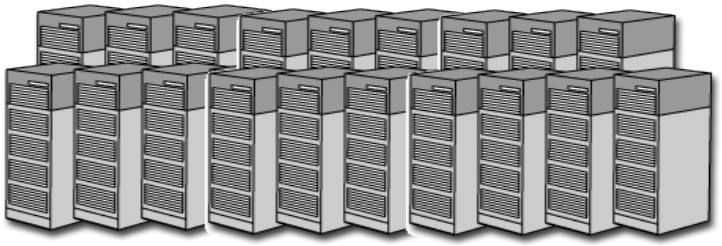
- Improves throughput - > competitive advantage
- Makes computers work harder - > financial advantage

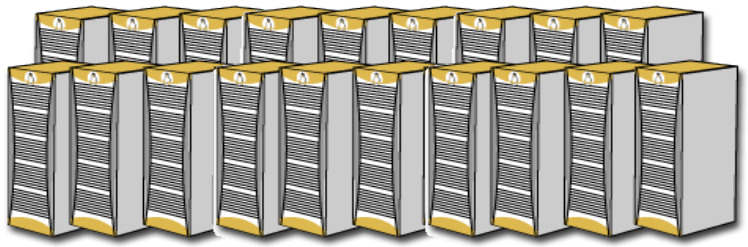
What it Does, How it Works

USER A

I need access to 100 computer systems for 48 hours to run a series of regression tests involving 600,000 test cases

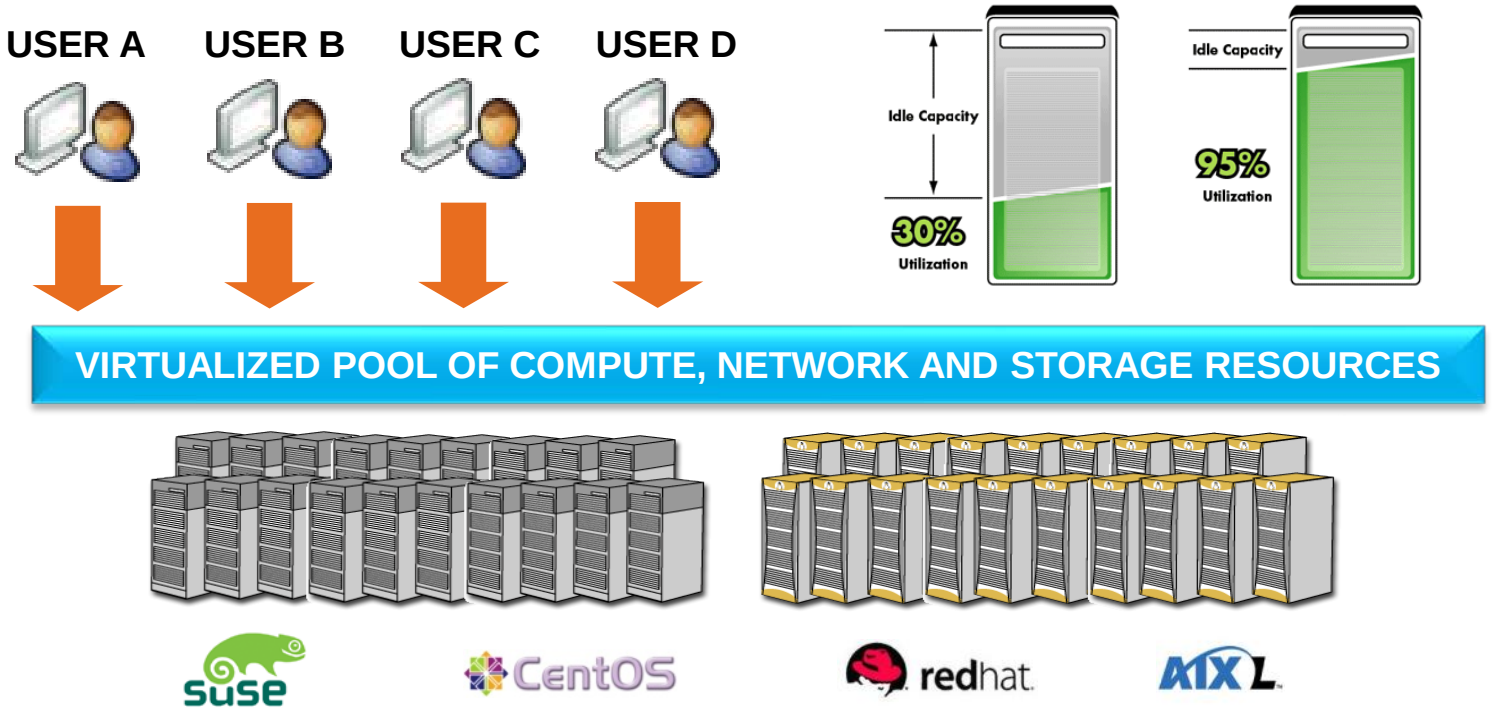
VIRTUALIZED POOL OF COMPUTE, NETWORK AND STORAGE RESOURCES





IBM Spectrum LSF

When users run applications under LSF they run more quickly, utilization is better, and applications are more reliable.



IBM Spectrum LSf can:

- Monitors systems and tracks dynamic resources (memory, swap & CPU) as well as static resources (OS, version, binary architecture, hardware).
- Selects the right system at the right time to run jobs held in configurable system wide queues.
- Ensures the reliable execution of jobs, re-starting them or migrating them as necessary in the case of hardware, network or software failures.
- Optimizes the allocation of various resources by user, project team or site, based on flexible policies that reflect business needs.

The LSF job script

- the user is required to provide a job script (also called batch file) with the specification of:
  - the resources requested, and the job constraints;
  - specific queue specification (more details later);
  - all what is needed for a working execution-environment.
- By resources we mean the number of **processors/cores/nodes**, the amount of **memory** needed (total and/or per process), and eventually some specific features (special hardware, for example), and so on.
- All these information must be written in a text file

LSF Script example

- For every line of resource specifications, #BSUB directive must be the first text of the line, and all specifications must come before any executable lines

```
#!/bin/bash
#
#BSUB –J jobname
#BSUB –n number-of-slots
#BSUB –o %J.stdout
#BSUB –e %J.stderr
#BSUB –R “span[ptile=40]”
#BSUB –q medium

module load mpi/intel/2018.4
module load app/version

mpirun app.exe
```

- submit it by typing in a terminal the command:
  - bsub < job.lsf

Check job status

- Then you can check the status of your submission issuing the command:
- bjobs
- bjobs –l
- bjobs -W 99956
- bjobs -WL 99956

Basic Job Specifications

- These basic options are:

Basic LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Job Name	-J [SomeText]	-J MyJob1	Set the job name to "MyJob1"
Shell	-L [Shell]	-L /bin/bash	Uses the bash shell to initialize the job's execution environment.
Wall Clock Limit	-W [hh:mm]	-W 24:15	Set wall clock limit to 24 hour 15 min
Core count	-n ##	-n 400	Assigns 400 job cores.
Cores per node	-R "span[ptile=##]"	-R "span[ptile=40]"	Request 40 cores per node.
Memory Per Core	-M [##]	-M 2GB	Sets the per process memory limit to 2GB.
Memory Per Core	-R "rusage[mem=[##]]"	-R "rusage[mem=2GB]"	Schedules job on nodes that have at least 2GBs available per core.
Combined stdout and stderr	-o [OutputName].%j	-o stdout1.%j	Collect stdout/err in stdout.[JobID]

Optional Job Specifications

- A variety of optional specifications is available to customize your job. The table below lists the specifications, which are most useful for users.

Optional LSF Job Specifications			
Specification	**Option**	**Example**	**Example-Purpose**
Set Allocation	-P #######	-P projectA	Set allocation to charge to Project projectA
Specify Queue	-q [queue]	-q short	Request only nodes in short subset.
Exclusive Node Usage	-x		Assigns a whole node exclusively for the job.
Specific node type	-R "select[gpu|phi]"	-R "select[gpu]"	Requests a node with a GPU to be used for the job.

Environment Variables

- Below is a list of most commonly used environment variables

Variable	Usage	Desc
Job ID	$LSB_JOBID	Batch job ID assigned by LSF.
Job Name	$LSB_JOBNAME	The name of the Job.
Queue	$LSB_QUEUE	The name of the queue the job is dispatched from.
Error File	$LSB_ERRORFILE	Name of the error file specified with a bsub -e.
Submit Directory	$LSB_SUBCWD	The directory the job was submitted from.
Hosts I	$LSB_HOSTS	The list of nodes that are used to run the batch job, repeated according to ptile value. *The character limit of LSB_HOSTS variable is 4096.
Hosts II	$LSB_MCPU_HOSTS	The list of nodes and the specified or default ptile value per node to run the batch job.
Host file	$LSB_DJOB_HOSTFILE	The hostfile containing the list of nodes that are used to run the batch job.

To see all relevant LSF environment variables for a job, add `env | grep LSB` to jobfile and submit job

Executable Commands

- After the resource, specification section of a job file comes the executable section.
- This executable section contains all the necessary UNIX, Linux, and program commands that will be run in the job.
- Some commands that may go in this section include, but are not limited to:
  - Changing directories
  - Loading, unloading, and listing modules
  - Launching software

Note:

- Remember that the UNIX is case sensitive and so uppercase and lowercase letters have different meanings, even in the job script specifications.
- Avoid using special characters, letters and spaces in the name of files, directories, and the job name. The script could be interpreted by the scheduler in unexpected ways.
- The default behaviour in LSF is to append the content of the file. So if you don't use unique names (like in the example), subsequent runs will add to the same file. To change the default behaviour to overwrite, use the flags -oo and -eo.
- If you do not specify the error file but only the output one, the standard output and standard error streams are merged and saved into the output file.
- The filename after the -o, -e flag can be a full path. If it is the name of an existing directory (ending with "/"), then LSF will automatically save as file with a name <jobid>.out.
- If you don't specify to save the output file somewhere, the content of the standard output and error streams will be appended to the report that is sent after completion (up to a system-predefined maximum size).

Manage LSF Jobs

- LSF Job Monitoring and Control Commands

Function	Command	Example
Submit a job	bsub < [script_file]	bsub < MyJob.LSF
Cancel/Kill a job	bkill [Job_ID]	bkill 101204
Check summary status of a single job	bjobs [job_id]	bjobs 101204
Check summary status of all jobs for a user	bjobs -u [user_name]	bjobs -u User1
Check detailed status of a single job	bjobs -l [job_id]	bjobs -l 101204
Modify job submission options	bmod [bsub_options] [job_id]	bmod -W 2:00 101204
Holding a batch job	bstop [jobid]	bstop 101204
Releasing a batch job	bresume [jobid]	bresume 101204

```
$ man [command]
```

LSF Job states

- The basic job states are:
  - **PEND** - the job is in the queue, waiting to be scheduled
  - **PSUSP** - the job was submitted, but was put in the suspended state (ineligible to run)
  - **RUN** - the job has been granted an allocation. If it’s a batch job, the batch script has been run
  - **DONE** - the job has completed successfully
- A pending job can remain pending for a number of reasons
  - **Dependency** - the pending job is waiting for another job to complete
  - **Priority** - the job is not high enough in the queue
  - **Resources** - the job is high in the queue, but there are not enough resources to satisfy the job’s request

MPI Job Templates

```
#!/bin/bash
#
#BSUB -J MPIJob                 ### set the job Name
#BSUB -q short                  ### specify queue
#BSUB -n 40                     ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”       ### ask for 40 cores per node
#BSUB -W 10:00                  ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out          ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err          ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute
# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

### This uses the LSB_DJOB_NUMPROC to assign all the cores reserved
### This is a very basic syntax. For more complex examples, see the documentation
mpirun -np $LSB_DJOB_NUMPROC ./MPI_program
```

OpenMP Jobs under LSF

```
#!/bin/bash
#
#BSUB -J OpenMPjob  ### set the job Name
#BSUB -q short                 ### specify queue
#BSUB -n 40                    ### ask for number of cores (default: 1)
#BSUB -R "span[hosts=1]"       ### specify that the cores MUST BE on a single host!
#BSUB -W 16:00                 ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out         ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err         ### -o and -e mean append, -oo and -eo mean overwrite

# set OMP_NUM_THREADS _and_ export!
OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
export OMP_NUM_THREADS

# Program_name_and_options
./openmp_program [options]
```

If you make use of special environment variables for your OpenMP program, remember to put them in your script (use the same syntax as the `OMP_NUM_THREADS` line in the script).

Hybrid MPI/OpenMP jobs under LSF

- MPI and OpenMP can be combined to write programs that exploit the possibility given by multi-core SMP machines, and the possibility of using the Message Passing Interface based communications between nodes on the cluster

```
#!/bin/bash
#
#BSUB –J Hybrid_MPI_OpenMP          ### set the job Name
#BSUB -q short                      ### specify queue
#BSUB -n 400                        ### ask for number of cores (default: 1)
#BSUB –R “span[ptile=40]”           ### ask for 40 cores per node
#BSUB -W 10:00                      ### set walltime limit: hh:mm
#BSUB -o stdout_%J.out              ### Specify the output and error file. %J is the job-id
#BSUB -e stderr_%J.err              ### -o and -e mean append, -oo and -eo mean overwrite

# here follow the commands you want to execute
# load the necessary modules
# NOTE: this is just an example, check with the available modules
module load intel/2018.4
module load mpi/intel/2018.4

# -- OpenMP environment variables --
Num=$(echo $LSB_SUB_RES_REQ | sed -e 's/span\[ptile=/ /g' -e 's/\]/ /g')
OMP_NUM_THREADS=$Num
export OMP_NUM_THREADS

mpirun -npernode 1 ./mpi_bybrid_program
```

Job Dependencies under LSF

- There may be situations where there are obvious dependencies between jobs, and one cannot start before another has finished(for example because it is dependent on the results of the previous jobs).
- LSF provides a dependency feature that can be used for this purpose. One can therefore specify in the batch script of a job, that this job has a dependency on another one. This is done with the syntax
- #BSUB -w "done(<jobid>)"

- This will ensure that the current job is started only after the job with the corresponding jobid is completed successfully. Some of the most common options are
  - **done** : job state done
  - **ended** : job state exit or done
  - **exit** : job state is EXIT and eventually with a specific exit code exit(<jobid>,<exit-code>)
  - **started** : job state DONE, EXIT, USUSP, SSUSP or RUN

- To know the jobs that a job depends on, type: bjdepinfo <jobid>
- To show the jobs that depend on a job, use the -c option: bjdepinfo -c <jobid>

Recommended Settings for Large Jobs

- For jobs larger than 2000 cores (50+ nodes), the following MPI settings are recommended to reduce the MPI startup time:

```
export I_MPI_HYDRA_BOOTSTRAP=lsf
export I_MPI_HYDRA_BRANCH_COUNT=XXX
export I_MPI_LSF_USE_COLLECTIVE_LAUNCH=1
```

- The XXX number should match the number of nodes (#Nodes = #Cores / #CoresPerNode) that your job will request.

Batch Job Translation Guide

- We provide some basic informations for the purposes of moving from SLURM/PBS/Torque to LSF

User Commands	LSF	Slurm	PBS/Torque
Job submission	bsub [script_file]	sbatch [script_file]	qsub [script_file]
Job deletion	bkill [job_id]	scancel [job_id]	qdel [job_id]
Job status (by job)	bjobs [job_id]	squeue --job [job_id]	qstat [job_id]
Job status (by user)	bjobs -u [user_name]	squeue -u [user_name]	qstat -u [user_name]
Queue list	bqueues	squeue	qstat -Q

Environment Variables	LSF	Slurm	PBS/Torque
Job ID	$LSB_JOBID	$SLURM_JOBID	$PBS_JOBID
Submit Directory	$LSB_SUBCWD	$SLURM_SUBMIT_DIR	$PBS_O_WORKDIR

Job Specifications

The table below lists various directives that set characteristics and resources requirements for jobs

Job Specification	LSF	Slurm	PBS/Torque
Script directive	#BSUB	#SBATCH	#PBS
Node Count	N/A (Calculated from: CPUs/CPUs_per_node)	-N [min[-max]]	-l nodes=[count]
CPUs Per Node	-R "span[ptile=count]"	--ntasks-per-node=[count]	-l ppn=[count]
CPU Count	-n [count]	-n [count]	N/A (Calculated from: CPUs_per_node * Nodes)
Wall Clock Limit	-W [hh:mm]	-t [min] or -t[days-hh:mm:ss]	-l walltime=[hh:mm:ss]
Memory Per Core	-M [mm] AND -R "rusage[mem=mm]" * Where mm is in MB	--mem-per-cpu=mem[M/G/T]	-l mem=[mm] * Where mm is in MB
Standard Output File	-o [file_name]	-o [file_name]	-o [file_name]
Standard Error File	-e [file_name]	-e [file_name]	-e [file_name]
Combine stdout/err	(use -o without -e)	(use -o without -e)	-j oe (both to stdout) OR -j eo (both to stderr)
Event Notification	-B and/or -N * For Begin and/or End	--mail-type=[ALL/END] * See 'man sbatch' for all types	-m [a/b/e] * For Abort, Begin, and/or End
Email Address	-u [address]	--mail-user=[address]	-M [address]
Job Name	-J [name]	--job-name=[name]	-N [name]
Account to charge	-P [account]	-account=[account]	-l billto=[account]
Queue	-q [queue]	-q [queue]	-q [queue]

thanks.

Different is better