HLRE-3 (mistral) Introduction

Hendryk Bockelmann
Deutsches Klimarechenzentrum (DKRZ)

Comparison of mistral (phase 1) and blizzard

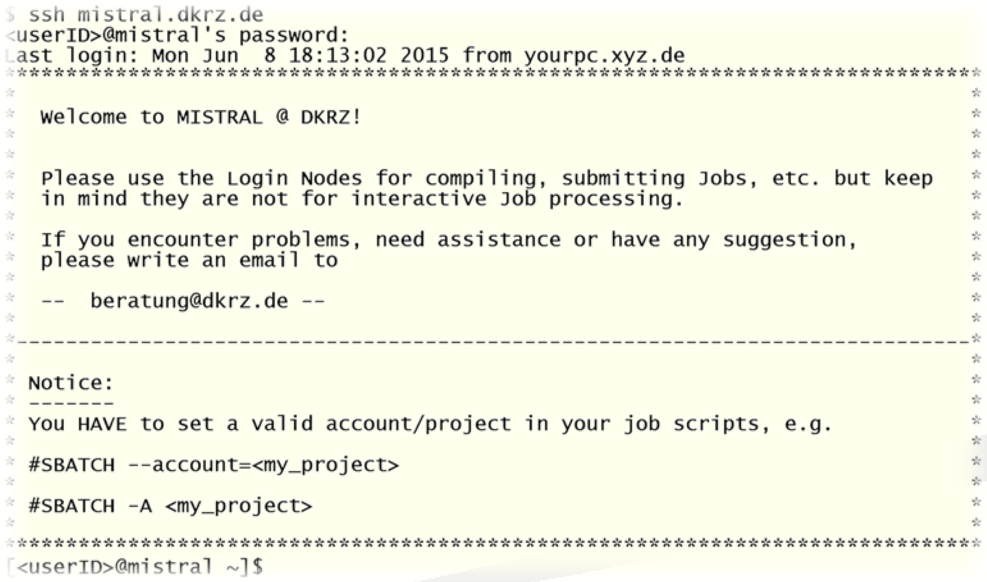
System	# nodes	Mem(TB)	# cores	TFLOP	Disk storage
blizzard	264	20	8448	158 peak	7 PByte
mistral (phase 1) total	**1556**	**115**	**37344**	**1493 peak**	**20 PByte**
compute (1386 x 64GB, 110 x 128GB)	1496	100	35904	1436 peak	
prepost (256GB)	48	12	1152	46 peak	
visualization (256GB + 2x Nvidia Tesla K80 GPUs)	12	3	288	11 peak (w/o GPUs)	

- Power6 node (32 cores): 4 DP flop/cycle at 4.7 GHz (2 FMA instructions)
  => 601.6 GFlop/s per node
- HSW node (24 cores): 16 DP flop/cycle at 2.5 GHz (2 x 256-bit FMA instr. – AVX2)
  => 960.0 GFlop/s per node (806.4 GFlop/s truly)

Configuration

- bullx B700 DLC (Direct Liquid Cooling) blade system with two nodes forming one blade
- each node has two sockets, equipped with an Intel Xeon E5-2680 v3 12-core processor (Haswell)
- four kinds of nodes are available to users:
  - 8 login nodes
  - 1496 compute nodes for running scientific models (thin & big memory)
  - 48 nodes for interactive use and pre-/postprocessing (fat memory)
  - 12 visualization/GPU nodes
- all nodes are integrated in one FDR Infiniband Fabric with fat tree topology (measured 5.9 GB/s bandwidth, 2.7µs latency)
- operating System is Red Hat Enterprise Linux release 6.4

Login and Environment



```
$ ssh mistral.dkrz.de
<userID>@mistral's password:
Last login: Mon Jun  8 18:13:02 2015 from yourpc.xyz.de
*****************************************************************************
*                                                                           *
*  Welcome to MISTRAL @ DKRZ!                                               *
*                                                                           *
*                                                                           *
*  Please use the Login Nodes for compiling, submitting Jobs, etc. but keep *
*  in mind they are not for interactive Job processing.                     *
*                                                                           *
*  If you encounter problems, need assistance or have any suggestion,       *
*  please write an email to                                                 *
*                                                                           *
*  --   beratung@dkrz.de --                                                 *
*                                                                           *
*---------------------------------------------------------------------------*
*                                                                           *
* Notice:                                                                   *
* -------                                                                   *
* You HAVE to set a valid account/project in your job scripts, e.g.         *
*                                                                           *
* #SBATCH --account=<my_project>                                            *
*                                                                           *
* #SBATCH -A <my_project>                                                   *
*                                                                           *
*****************************************************************************
[<userID>@mistral ~]$
```

Login and Environment

- users need to be member in at least one active HLRE project – all blizzard users should have access to mistral
- remember: you have to update your user account information at https://luv.dkrz.de and accept the 'guidelines for use'
- log into mistral via ssh, replacing <userid> by your username:

```
ssh <userid>@mistral.dkrz.de
```

- mistral.dkrz.de is a round robin to one of the login nodes: **mlogin100-107**
- login nodes serve as a frontend to the compute nodes of the HPC cluster
  - file editing and compilation of source code
  - submission, monitoring and canceling of batch jobs
  - can only be used for simple and not too time- and memory-intensive operations; otherwise use pre-/postprocessing nodes
- all DKRZ systems are manged by the LDAP protocol: change password and/or login shell through https://luv.dkrz.de

Software Environment

- as on blizzard the modules environment is used
- no hierarchical modules just naming convention

  <modname>/<modversion>
- internal consistency checks should warn if modules are incompatible
- **no** modules are loaded by default – hence, no "default compiler" (unless system gcc …)

Software Environment

Management via module sub-commands:

- module avail: show list of all available modules
- module add: load a specific module – use full description <modname>/<modversion> or latest version <modname>
- module list: list currently loaded modules
- module rm <modname>/<modversion> : unload module
- module purge: unload all modules

Software Environment

- use of modules needed for whole workflow, e.g. pftp, autoconf, automake, … do not rely on /usr/bin
- but: not all software is made accessible via modules (mostly only those which change $PATH)
- e.g. NetCDF has no module (should we change this?), instead explore

  /sw/rhel6-x64

- alternatively use softwarelist at

  https://www.dkrz.de/Nutzerportal-en/doku/mistral/softwarelist

Future version will contain information whether module is present or absolute path is given …

Software Environment

- Compiler:
  - intel (to be used for production), gcc, nag (for debugging)
  - ask Michael after lunch for best settings of intel compiler …
- MPI:
  - bullxMPI (with mellanox tools), IntelMPI, mvapich2 and openmpi (no further support, just for testing)
  - ask Cyril in the next session for best settings of bullxMPI gained from the benchmarks of the procurement …

Data Management – Filesystem

Data Management – Filesystem

- parallel filesystem lustre with 3 data spaces as on blizzard
  - HOME
  - WORK
  - SCRATCH
- all data spaces are available on all nodes
- no differentiation between small and big files needed (cmp. blocksize on GPFS)

Data Management – Filesystem

system	HOME	WORK	SCRATCH
Path	/pf/[a,b,g,k,m,u]/<userid>	/work/<projectid>	/scratch/[a,b,g,k,m,u]/<userid>
Description	- Assigned to user account - Storage of personal files, src code, etc	- Assigned to project account - Interim storage of output from experiments and frequently used data	- Assigned to user account - Temporary storage and processing of large data sets
Quota	24 GB	According to annual project allocation	15 TB
Backup	Yes	No	No
Data lifetime	Until user account deletion	1 month after project expiration	14 days since last file access

Data Management – Filesystem

- old data on blizzard GPFS is mirrored for you – only HOME, WORK and /pool/data
- find data at: /mnt/lustre01/rsync/[pf|work]
- /pool/data and HOME synchronized daily
- last rsync of data between GPFS and lustre is done on **01.08.2015 (planned – there will be an announcement)**
- do not use blizzard GPFS afterwards or copy/sync yourself …
- meanwhile cp from /mnt/lustre01/rsync; **NO** mv from /mnt/lustre01/rsync/ or scp from blizzard, please!

Lustre

I/O architecture

- File system: lustre 2.5
- 29 I/O server and 5 metadata server
- max performance per server is 5.4 GiB/s

Best Practices

- lustre striping parameters might influence I/O performance
- Set number of stripes: ‘lfs setstripe –c 4 /work/k10100/k101042/lfs_test’
- Default for all directories is 1: ‘lfs getstripe –c $HOME’
- Reasonable values for large (>> GiB) shared files are: 4 to 16
- Never set the value too high as the slowest server is the limitation
- Remember: it is not trivial to achieve good performance

$\Rightarrow$ Contact DKRZ if you have issues with I/O

Every day work and scripting

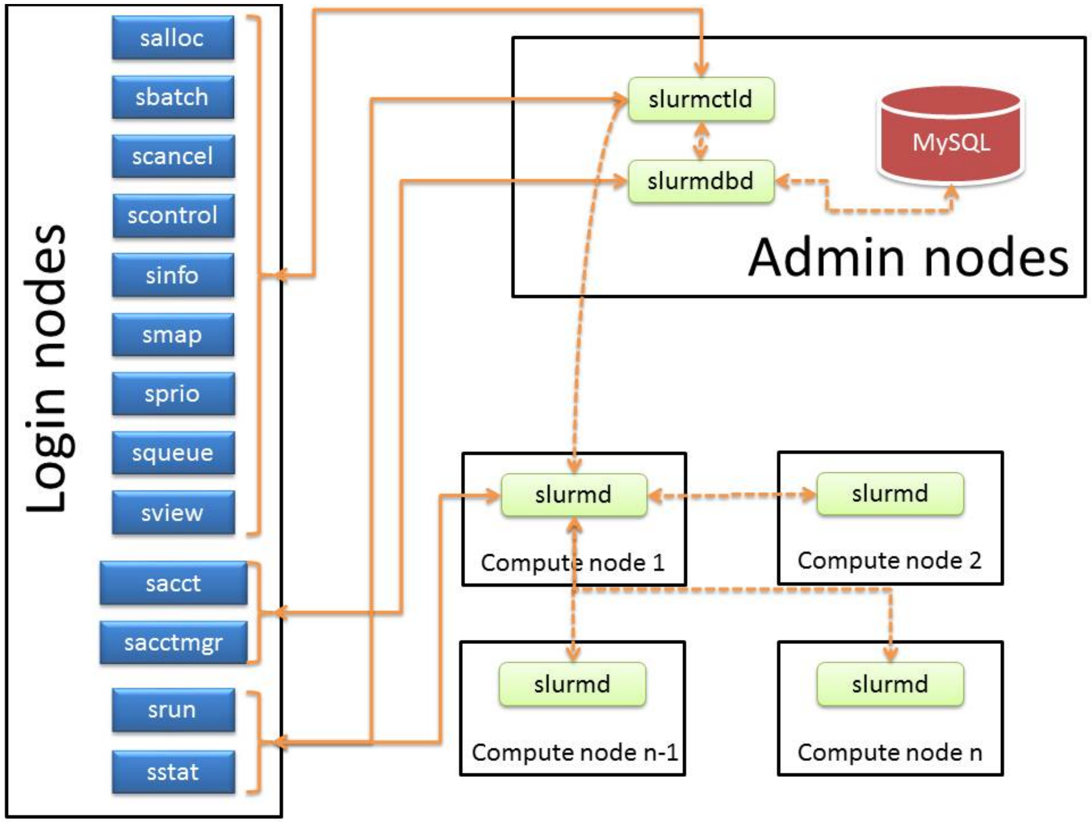
- use lfs commands for faster response!

  e.g. ‘lfs ls –l’ instead of ‘ls –l’

SLURM scheduler

SLURM Entities

Simple Linux Utility for Resource Management



SLURM Resource Management

- Partitions and QoS (Quality of Service) are used in SLURM to group nodes and jobs characteristics
- The use of Partitions and QoS entities in SLURM is orthogonal:
  - Partitions for grouping resources characteristics
  - QoS for grouping limitations and priorities

Partition *compute*: 512 nodes max, 8h runtime

Partition *gpu*: 2 nodes max, Nvidia GPU

Partition *prepost*: 2 nodes max, high memory

QoS *express*: higher priority, 20 min runtime limit, 4 nodes max

QoS long: lower priority, higher runtime

SLURM Partitions

Partition	compute (default)	prepost	shared	gpu
MaxNodes	512	2	1	2
MaxTime	8 hours	4 hours	7 days	4 hours
Shared	exclusive	yes:4 max 4 jobs share resources	yes:4 max 4 jobs share resources	to be defined, replacement for halo
MaxMemPerCPU	nodelimit	10 Gbyte	2.5 GByte	10 Gbyte

General limits:

- 20 jobs running in parallel (no limit for queued jobs)
- might be extended

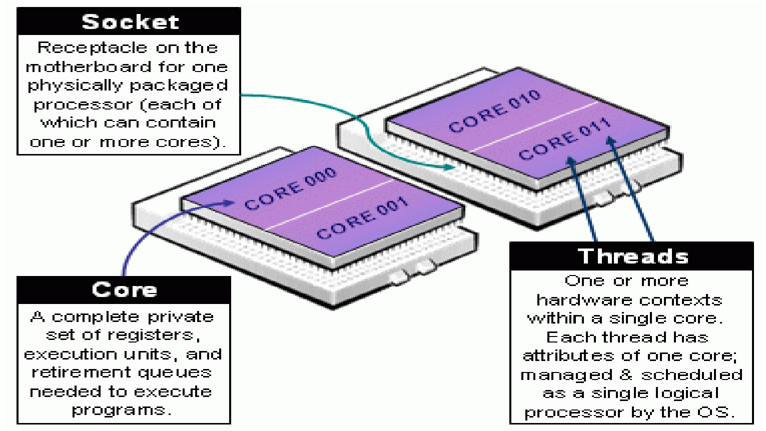
SLURM Associations

```
$ sacctmgr list user format=user,defaultaccount where user=$USER
      User   Def Acct
---------- ----------
   k202082  noaccount
$ sacctmgr list associations format=user,account,partition,qos where
user=$USER
      User    Account  Partition                  QOS
---------- ---------- ---------- --------------------
   k202082  noaccount                          normal
   k202082     k20200           bench,express,normal
   k202082     bm0834                 express,normal
```

- you have to specify the account for each job !
- list partitions: ‘scontrol show partitions’
- list qos: ‘sacctmgr show qos’

SLURM Entities

- **partition** : job queue with limits and access controls
- **job** : request for resource allocation
- **job-step** : set of (typically parallel) tasks
- **node** : NUMA board
  - 2 sockets with 12 cores each
    - each core 2 CPUs/HyperThreads
  - Memory
  - generic resources (like GPUs)


Socket
Receptacle on the motherboard for one physically packaged processor (each of which can contain one or more cores).
CORE 010
CORE 011
CORE 000
CORE 001
Core
A complete private set of registers, execution units, and retirement queues needed to execute programs.
Threads
One or more hardware contexts within a single core. Each thread has attributes of one core; managed & scheduled as a single logical processor by the OS.

SLURM Job and Node States

- Job States (squeue)
  - **Pending PD** (awaiting resources allocation)
  - **Running R** (has an active allocation)
  - **Cancelled CA** (explicitly cancelled by user or admin)
  - **Timeout TO** (terminated reaching time limit)
  - **Completing CG** (some nodes may still be active)
  - **Completed CD** (all processes on all nodes terminated)

- Node States (sinfo)
  - **Down** (unavailable for use)
  - **Idle** (not allocated, awaiting job allocations)
  - **Allocated** (by one or more jobs)
  - **Completing** (jobs are completing, epilog might run)
  - **Draining** (currently running job, but will not be available afterwards)
  - **Mixed** (some CPUs allocated while others idle)

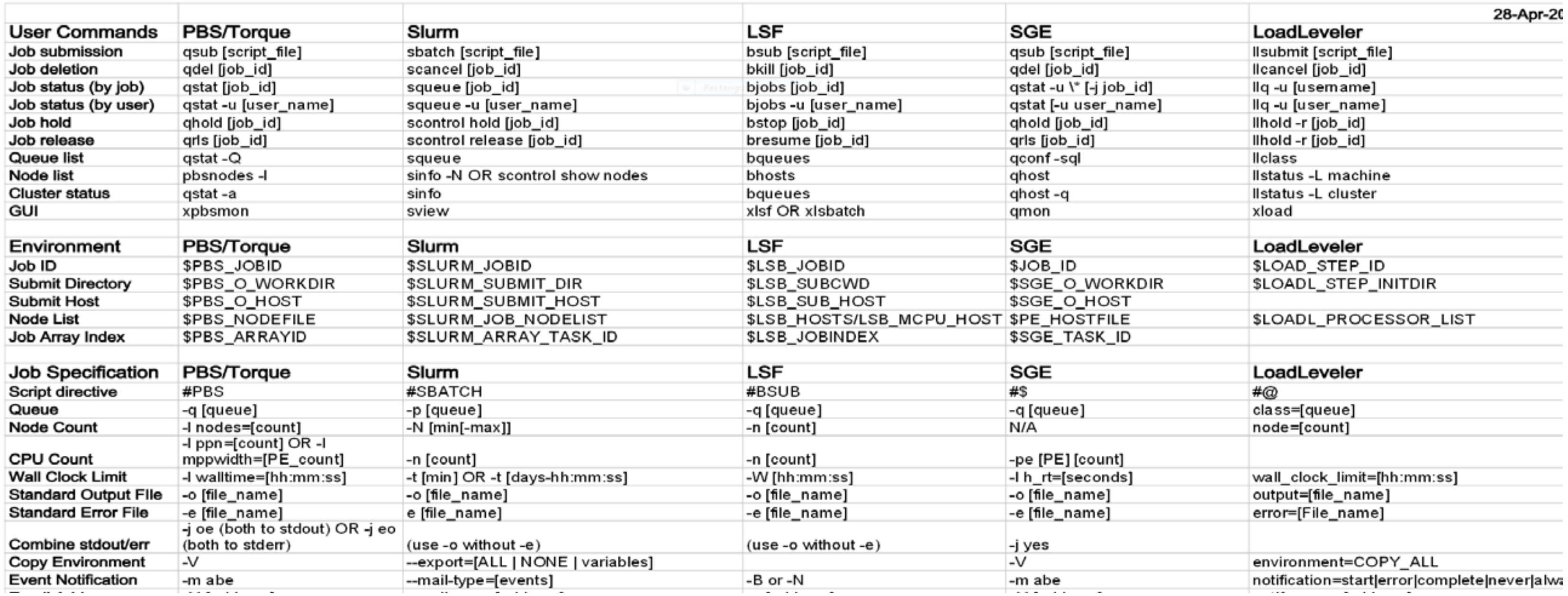
* after state means node is not reachable for slurmctld

SLURM Command Overview

- **sinfo** – show information about partition and nodes
- **squeue** – list pending and running jobs
- **sbatch** – submit job script for execution (batch mode)
- **salloc** – create job allocation and start a shell to use it (interactive mode)
- **srun** – create a job allocation and directly launch a job step (typically an MPI job)
- **scancel** – cancel pending or running job or job step
- **scontrol** – show or modify jobs, partitions, nodes, …

Side by side comparison to LoadLeveler

http://slurm.schedmd.com/rosetta.pdf



28-Apr-20

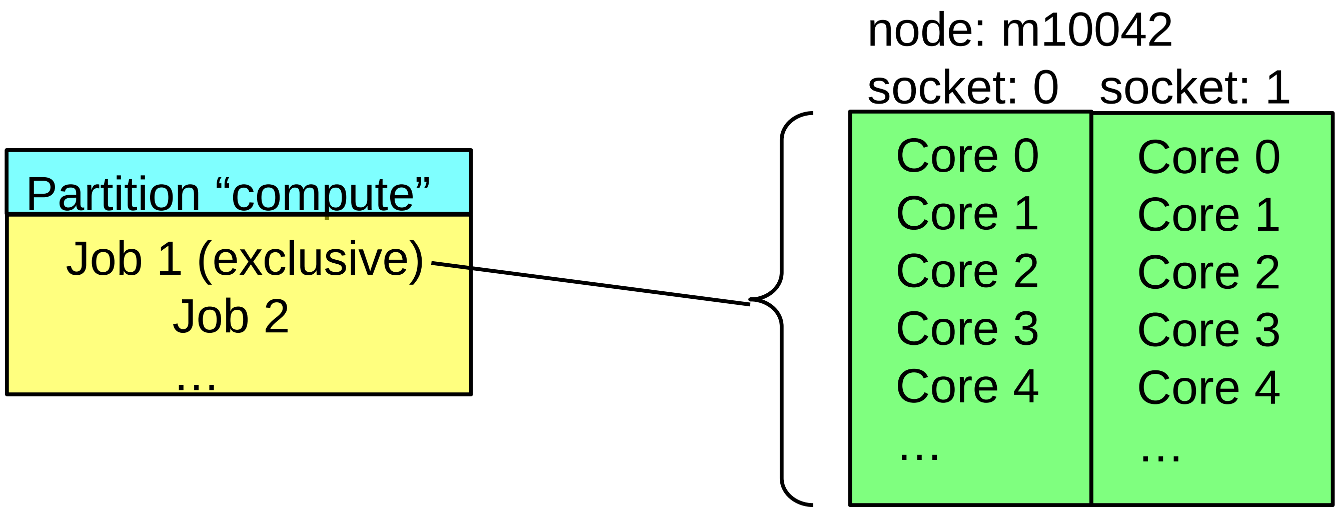
User Commands	PBS/Torque	Slurm	LSF	SGE	LoadLeveler
Job submission	qsub [script_file]	sbatch [script_file]	bsub [script_file]	qsub [script_file]	llsubmit [script_file]
Job deletion	qdel [job_id]	scancel [job_id]	bkill [job_id]	qdel [job_id]	llcancel [job_id]
Job status (by job)	qstat [job_id]	squeue [job_id]	bjobs [job_id]	qstat -u * [-j job_id]	llq -u [username]
Job status (by user)	qstat -u [user_name]	squeue -u [user_name]	bjobs -u [user_name]	qstat [-u user_name]	llq -u [user_name]
Job hold	qhold [job_id]	scontrol hold [job_id]	bstop [job_id]	qhold [job_id]	llhold -r [job_id]
Job release	qrls [job_id]	scontrol release [job_id]	bresume [job_id]	qrls [job_id]	llhold -r [job_id]
Queue list	qstat -Q	squeue	bqueues	qconf -sql	llclass
Node list	pbsnodes -l	sinfo -N OR scontrol show nodes	bhosts	qhost	llstatus -L machine
Cluster status	qstat -a	sinfo	bqueues	qhost -q	llstatus -L cluster
GUI	xpbsmon	sview	xlsf OR xlsbatch	qmon	xload
Environment	**PBS/Torque**	**Slurm**	**LSF**	**SGE**	**LoadLeveler**
Job ID	$PBS_JOBID	$SLURM_JOBID	$LSB_JOBID	$JOB_ID	$LOAD_STEP_ID
Submit Directory	$PBS_O_WORKDIR	$SLURM_SUBMIT_DIR	$LSB_SUBCWD	$SGE_O_WORKDIR	$LOADL_STEP_INITDIR
Submit Host	$PBS_O_HOST	$SLURM_SUBMIT_HOST	$LSB_SUB_HOST	$SGE_O_HOST	
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST	$LSB_HOSTS/LSB_MCPU_HOST	$PE_HOSTFILE	$LOADL_PROCESSOR_LIST
Job Array Index	$PBS_ARRAYID	$SLURM_ARRAY_TASK_ID	$LSB_JOBINDEX	$SGE_TASK_ID	
Job Specification	**PBS/Torque**	**Slurm**	**LSF**	**SGE**	**LoadLeveler**
Script directive	#PBS	#SBATCH	#BSUB	#$	#@
Queue	-q [queue]	-p [queue]	-q [queue]	-q [queue]	class=[queue]
Node Count	-l nodes=[count]	-N [min[-max]]	-n [count]	N/A	node=[count]
CPU Count	-l ppn=[count] OR -l mppwidth=[PE_count]	-n [count]	-n [count]	-pe [PE] [count]	
Wall Clock Limit	-l walltime=[hh:mm:ss]	-t [min] OR -t [days-hh:mm:ss]	-W [hh:mm:ss]	-l h_rt=[seconds]	wall_clock_limit=[hh:mm:ss]
Standard Output File	-o [file_name]	-o [file_name]	-o [file_name]	-o [file_name]	output=[file_name]
Standard Error File	-e [file_name]	e [file_name]	-e [file_name]	-e [file_name]	error=[File_name]
Combine stdout/err	-j oe (both to stdout) OR -j eo (both to stderr)	(use -o without -e)	(use -o without -e)	-j yes	
Copy Environment	-V	--export=[ALL | NONE | variables]		-V	environment=COPY_ALL
Event Notification	-m abe	--mail-type=[events]	-B or -N	-m abe	notification=start|error|complete|never|alwa

http://slurm.schedmd.com/pdfs/summary.pdf

SLURM Workflow

Example to demonstrate job allocations and job steps:

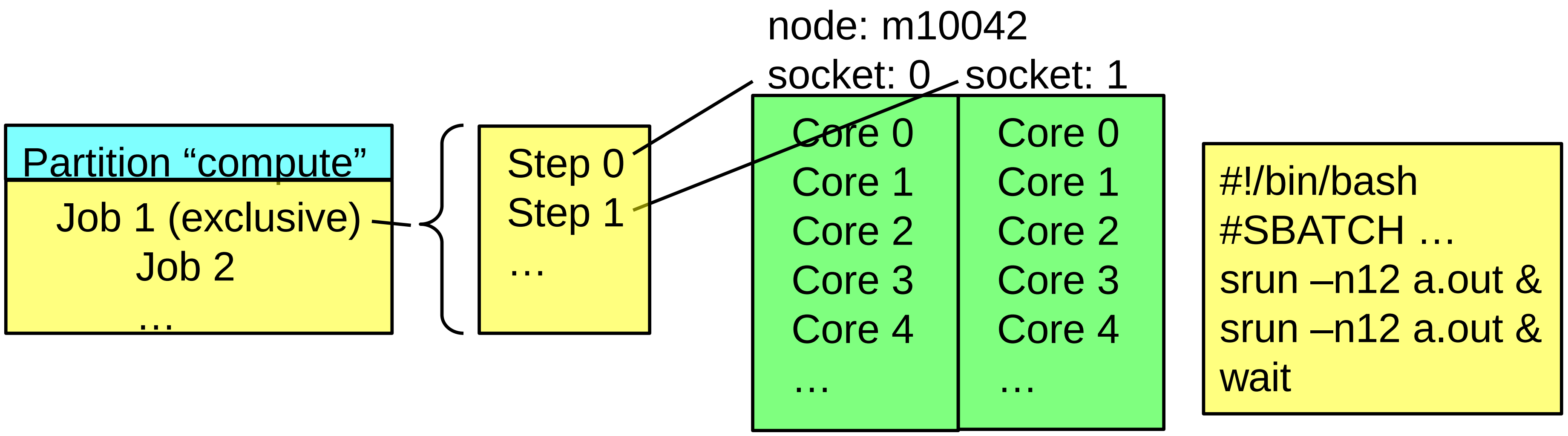
- Jobs are allocated resources (sbatch or salloc)



SLURM Workflow

Example to demonstrate job allocations and job steps:

- Jobs spawn job steps, which are allocated resources from within the job's allocation (srun)



```
#!/bin/bash
#SBATCH …
srun –n12 a.out &
srun –n12 a.out &
wait
```

SLURM Interactive Usage

```
$ salloc -p prepost -N2 -n2 -Ak20200
salloc: Granted job allocation 125037
$ env | grep -i slurm
SLURM_NODELIST=m[11512-11513]
SLURM_NNODES=2
SLURM_JOBID=125037
SLURM_NTASKS=2
SLURM_TASKS_PER_NODE=1(x2)
SLURM_SUBMIT_HOST=mlogin103
…
$ hostname
mlogin103
$ srun --label hostname
0: m11512
1: m11513
```

ask for 2 tasks on 2 node

Environment Variables set

Subshell executed on submit host

To execute commands on compute nodes use srun

SLURM Interactive Usage

```
$ srun --label bash
hostname
0: m11512
1: m11513
env | egrep "SLURM_(JOBID|NODEID|PROCID|STEPID)"
0: SLURM_JOBID=125037
0: SLURM_STEPID=1
0: SLURM_NODEID=0
0: SLURM_PROCID=0
1: SLURM_JOBID=125037
1: SLURM_STEPID=1
1: SLURM_NODEID=1
1: SLURM_PROCID=1
exit
$ hostname
mlogin103
$ env | grep JOBID
SLURM_JOBID=125037
$ ssh -X m11512
$ hostname
m11512
$ exit
Connection to m11512 closed.
exit
salloc: Relinquishing job allocation 125037
```

To execute a shell for all allocated nodes

No Prompt is displayed!

Command line is executed on all nodes in parallel

Back on submit host
Still inside allocation

Login directly on compute node is possible

SLURM Spawning Tasks

- both sbatch and salloc only allocate resources
- use srun to start parallel (MPI) application
- for testing purposes one might call srun directly from login node:

```
$ srun –N2 –n2 –Ak20200 -l hostname
srun: job 125356 queued and waiting for resources
srun: job 125356 has been allocated resources
0: m11225
1: m11226
```

SLURM Batch Jobs

- submit job scripts using **sbatch** command
- shell script executed on the first node of the allocation
- SLURM submission options are prefixed with

  #SBATCH
- short or long option, e.g. --nodes=<n> or –N <n>
- for long form: **NO** space around = allowed
- #SBATCH ignored after first executable command in script
- options can also be given on command line and have highest priority

SLURM Batch Jobs

#SBATCH option	Default value	Description
--nodes=<n>	1	Number of nodes for the allocation
--ntasks=<n>	1	Number of (MPI) tasks
--ntasks-per-node=<n>	1	Number of (MPI) tasks per node
--cpus-per-task=<n>	1	Number of threads (logical CPUs) per task
--time=<walltime>	partition dependent	Requested wallclock limit
--partition=<name>	compute	Partition for the allocation
--account=<projectid>	NONE	Project Id for the accounting

SLURM Batch Jobs

Resource selection and resource allocation options can be mixed in sbatch …

- Selection:

```
#SBATCH --nodes=32
#SBATCH --cores-per-socket=12
```

- Allocation:

```
#SBATCH --ntasks=12
#SBATCH --ntasks-per-socket=4
#SBATCH --cpus-per-task=8
```

- srun inherits option from sbatch if not overwritten

CAUTION: HyperThreading

- mistral Haswell processors support HyperThreading (cmp SMT on blizzard)
- i.e. each node has 24 physical cores, 48 logical CPUs or Hardware Threads (HWT)
- SLURM offers option --threads-per-core to distinguish between nodes in a heterogeneous system
- BUT: on mistral --threads-per-core=2 is enforced! Even if you do not want to use HT
- Respect this when setting --cpus-per-task

CAUTION: Frequency Scaling

- mistral Haswell processors allow for CPU frequency scaling
- Operating system can scale CPU frequency up or down in order to save power
- default srun behaviour: use fixed 2.5 GHz
- be friendly to the environment and save energy ☺

  export SLURM_CPU_FREQ_REQ=1200000

  or

  srun --cpu-freq=1200000
- frequency has to be given in kiloHertz (kHz)
- try the impact of lower frequencies (1.2, 1.3, ..., 2.5 GHz) on the runtime of your code

Example: OpenMP job

```
#!/bin/bash
#SBATCH --job-name=OpenMPjob
#SBATCH --partition=shared
#SBATCH --ntasks=1
#SBATCH --cpus-per-tasks=8
#SBATCH --time=00:30:00
#SBATCH --account=x12345

export OMP_NUM_THREADS=8
export KMP_AFFINITY=verbose,granularity=thread,compact,1
export KMP_STACKSIZE=64m

cdo –P 8 <operator> <ifile> <ofile>
```

Example: MPI job

```
#!/bin/bash
#SBATCH --job-name=MPIjob
#SBATCH --partition=compute
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=24
#SBATCH --cpus-per-tasks=2
#SBATCH --time=00:30:00
#SBATCH --account=x12345

module load intelmpi
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

srun –l --cpu_bind=verbose,cores ./myapp
```

Example: MPI job with and w/o HT

```
#!/bin/bash
#SBATCH --job-name=MPIjob
#SBATCH --partition=compute
#SBATCH --nodes=4
#SBATCH --time=00:30:00
#SBATCH --account=x12345

module load intelmpi
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

# first run without HyperThreading
srun –l --cpu_bind=verbose --hint=nomultithread --ntasks-per-node=24 ./myapp

# second run with HyperThreading
srun –l --cpu_bind=verbose --hint=multithread --ntasks-per-node=48 ./myapp
```

SLURM MPI support

- many different MPI implementations are supported
  - IntelMPI
  - bullxMPI
  - MVAPICH2
  - OpenMPI
- always use srun to launch the tasks – **no** mpirun, mpiexec, mpiexec.hydra, …
- some MPI option might be set via environmental variables: e.g. I_MPI_DEBUG=4 for Intel MPI
- see DKRZ website and/or user’s guide for examples

SLURM MPMD support

- Different programs may be launched by task ID with different program arguments
- Use “--multi-prog” option and specify configuration file instead of executable program
- Configuration file lists task IDs, executable programs, and arguments (“%t” mapped to task ID and “%o” mapped to offset within task ID range)

```
> cat mpmd.conf
#TaskID  Program                          Arguments
0-23     /pf/z/z123456/test/ocean
24-47    /pf/z/z123456/test/atmos   --rank=%o

> srun –l --ntasks=48 --multi-prog mpmd.conf
```

SLURM Process and Thread Binding

- Mapping processes: distribution of ranks on nodes

  srun --distribution=<block|cyclic[:block|cyclic]>

  - first argument: block or cycle on successive nodes
  - second argument: block or cycle on successive sockets
- Binding ranks to cores, cpus, ...

  srun --cpu_bind=<[verbose,]type>

  - bind to a single core: <type> = cores
  - bind to a single CPU/HyperThread: <type> = threads
  - custom bindings: <type> = map_cpu:<list>

SLURM Process and Thread Binding

- Binding OpenMP threads to cores, cpus

  export KMP_AFFINITY=[<modifier>,...]<type>[,<permute>]

  - modifier
    - verbose : very helpful !
    - granularity=core : reserve full physical core
    - granularity=thread : reserve logical CPU/HyperThread
  - type
    - compact : place threads as close as possible
    - scatter : distribute threads as evenly as possible
  - permute : sets most significant level of topology map, i.e. 0=CPUs (default), 1=cores, 2=socket/LLC

SLURM Process and Thread Binding

For hybrid MPI/OpenMP combine both:

1 node, 6 MPI tasks and 4 OpenMP threads per task, no use of HyperThreading:

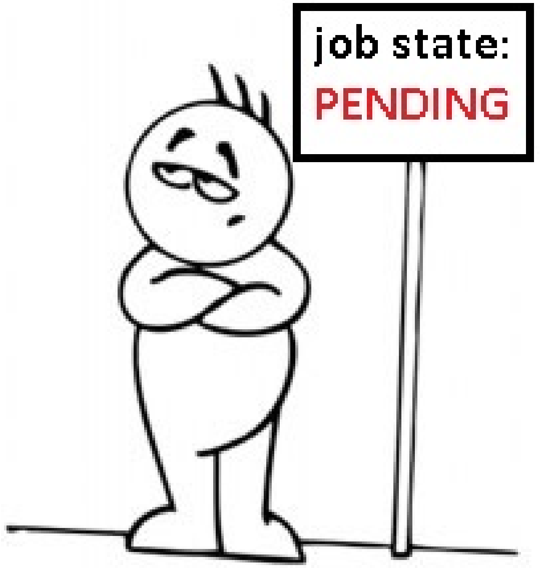
```
#SBATCH --tasks-per-node=6
#SBATCH --cpus-per-task=8
export OMP_NUM_THREADS=4
export KMP_AFFINITY=granularity=core,compact,1
srun –l --cpu_bind=cores ./myapp
```

SLURM Process and Thread Binding

Usage of HyperThreading allows to double e.g. number of MPI-tasks:

```
#SBATCH --tasks-per-node=12
#SBATCH --cpus-per-task=4
export OMP_NUM_THREADS=4
export KMP_AFFINITY=granularity=thread,compact,1
srun –l --cpu_bind=threads ./myapp
```

Solving Problems



SLURM allows to help yourself

- Simple postprocessing job script

```
$ cat test.job
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 2
#SBATCH -o /pf/z/z123456/test/job-%j.out
#SBATCH -e /pf/z/z123456/test/job-%j.err
#SBATCH -p prepost
#SBATCH –A x12345
srun –l hostname
export OMP_NUM_THREADS=2
cdo …
$ sbatch test.job
Submitted batch job 123321
```

SLURM allows to help yourself

- why is my job pending in the queue?

```
$ squeue -j 123321
         JOBID PARTITION     NAME     USER  ST        TIME  NODES NODELIST(REASON)
        123321   prepost test.job       me  PD        0:00      1 (ReqNodeNotAvail)
$ sinfo -t idle
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*       up   8:00:00    242   idle m[10000-10241]
prepost        up   4:00:00     39   idle m[11512-11517,11519-11531,11533-11553]
shared         up   infinite     0    n/a
gpu            up   4:00:00      0    n/a
```

SLURM allows to help yourself

- why is my job pending in the queue?

```
$ scontrol show job 123321
JobId=123321 Name=test.job
   UserId=me(2054944) GroupId=tests(200026)
   Priority=13022 Nice=0 Account=x12345 QOS=normal
   JobState=PENDING Reason=ReqNodeNotAvail Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=04:00:00 TimeMin=N/A
   SubmitTime=2015-07-01T06:32:11 EligibleTime=2015-07-01T06:32:11
   StartTime=2015-07-01T11:20:24 EndTime=Unknown
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=prepost AllocNode:Sid=mlogin103:2047
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null)
   NumNodes=1-1 NumCPUs=2 CPUs/Task=1 ReqS:C:T=*:*:*
   MinCPUsNode=1 MinMemoryNode=250G MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=0 Contiguous=0 Licenses=(null) Network=(null)
   Command=/pf/z/z123456/test/test.job
   WorkDir=/pf/z/z123456/test

$ scontrol show reservation
ReservationName=dkrz_42 StartTime=2015-07-01T13:00:00 EndTime=2015-07-01T18:00:00 Duration=05:00:00
   Nodes=m[11512-11559] NodeCnt=48 CoreCnt=2304 Features=(null) PartitionName=(null) Flags=SPEC_NODES
   Users=dkrz Accounts=(null) Licenses=(null) State=INACTIVE
```

SLURM allows to help yourself

- why is my job pending in the queue?

```
$ scontrol show job 123321
JobId=123321 Name=test.job
   UserId=me(2054944) GroupId=tests(200026)
   Priority=13022 Nice=0 Account=x12345 QOS=normal
   JobState=PENDING Reason=ReqNodeNotAvail Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=04:00:00 TimeMin=N/A
   SubmitTime=2015-07-01T06:32:11 EligibleTime=2015-07-01T06:32:11
   StartTime=2015-07-01T11:20:24 EndTime=Unknown
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=prepost AllocNode:Sid=mlogin103:2047
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null)
   NumNodes=1-1 NumCPUs=2 CPUs/Task=1 ReqS:C:T=*:*:*
   MinCPUsNode=1 MinMemoryNode=250G MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=0 Contiguous=0 Licenses=(null) Network=(null)
   Command=/pf/z/z123456/test/test.job
   WorkDir=/pf/z/z123456/test

$ scontrol show reservation
ReservationName=dkrz_42 StartTime=2015-07-01T13:00:00 EndTime=2015-07-01T18:00:00 Duration=05:00:00
   Nodes=m[11512-11559] NodeCnt=48 CoreCnt=2304 Features=(null) PartitionName=(null) Flags=SPEC_NODES
   Users=dkrz Accounts=(null) Licenses=(null) State=INACTIVE
```

SLURM allows to help yourself

- modify your request

```
$ cat test.job
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 2
#SBATCH -o /pf/z/z123456/test/job-%j.out
#SBATCH -e /pf/z/z123456/test/job-%j.err
#SBATCH -p prepost
#SBATCH –A x12345
#SBATCH --time=5
srun –l hostname
export OMP_NUM_THREADS=2
cdo …

$ scontrol update jobid=123321 timelimit=5
$ squeue –u $USER
 JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
123321   prepost test.job       me   R       0:02      1 m11513
```

Any questions?

If not: have fun working on mistral
If not now: contact beratung@dkrz.de

Documentation will be available at
https://www.dkrz.de/Nutzerportal-en/doku/mistral

Many thanks to Jan Wender and Fisnik Kraja from S&C for providing some slides.