Introduction to Linux and Supercomputers
Zhores Software (modules, batch)

MA030366 (Term 2) R2-B5-2026

Zacharov Igor

i.zacharov@skoltech.ru

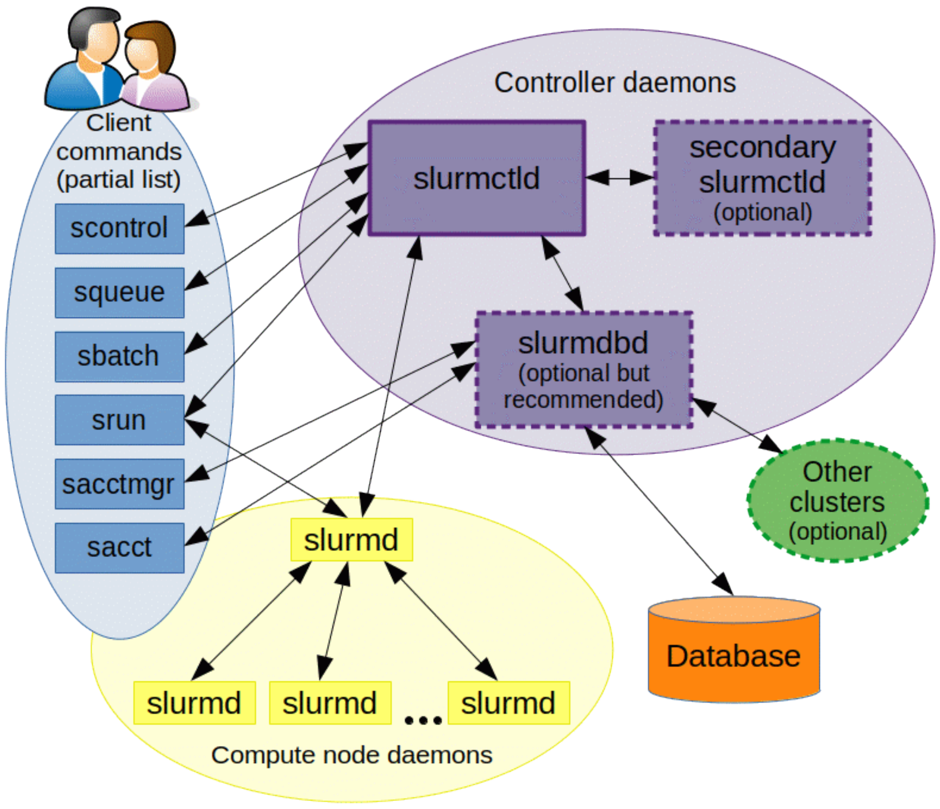
SLURM

Simple Linux Utility for Resource Management

- Job scheduling in cluster systems
  - Job: self-contained unit of work
    - Work may be described (automated) in a (bash) script
  - Scheduling: an upfront allocation of resources necessary to complete the job.
  - Scheduling may be based on:
    - Job priority
    - Estimated execution time
    - Resource requirements (number of CPU/GPU needed, size memory, etc.)
  - Job Queue: data structure maintained by Scheduler containing jobs to run
    - Submit job: action of copying job (description = script) in to the queue (data structure)
- Use of batch queue gives these benefits:
  - Sharing of computer resources among many users
  - Time-shift processing when resources are available
  - Avoids human supervision for scheduling
  - Allows around-the-clock utilization of computing resources
- Other such systems
  - NQE/PBS, Condor, LSF, SUN Grid Engine (SGE)

SLURM Queues Architecture

- Partition
  - Group of nodes with specific characteristic
  - Queue is organizing access of jobs for a partition
  - Job may have multiple steps, which are sets of tasks within a job
- Slurm does four basic steps to manage CPU resources for job/step:
  1. Selection of nodes
  2. Allocation of CPUs from selected nodes
  3. Distribution of tasks to selected nodes
  4. Optional binding of tasks to allocated CPUs
- One central controller daemon (slurmctld) on management node
  - Obtains resource utilization information
  - Makes scheduling decisions within partitions
- A daemon on each computing node (slurmd)
- One central daemon for the accounting database (slurmdbd)


Client commands (partial list)
scontrol
squeue
sbatch
srun
sacctmgr
sacct
Controller daemons
slurmctld
secondary slurmctld (optional)
slurmdbd (optional but recommended)
Other clusters (optional)
Database
slurmd
slurmd
slurmd
...
slurmd
Compute node daemons

SLURM Job status change

- SLURM provides the Final-State machine for changing the job status
  - At submission it is "PENDING" (PD) – waiting for the resources
  - When resources are allocated the job is "RUNNING" (R)
  - The status is given in the output of the squeue [-l] command

Status	**Code**	**Explanation**
COMPLETED	CD	Job has completed
COMPLETING	CG	Job is finishing but some processes are still active
FAILED	F	Job terminated with a non-zero exit code (failed)
PENDING	PD	Job is waiting for resource allocation
PREEMPTED	PR	Job was terminated because of preemption by another job
RUNNING	R	Job currently is allocated to a node and running
SUSPENDED	S	Job has been stopped with cores released to another job
STOPPED	ST	Job has been stopped with its cores retained

```
JOBID     PARTITION      NAME     USER     STATE          TIME  TIME_LIMI  NODES  NODELIST(REASON)
1169972   gpu_devel      bash    i.zachar  COMPLETING     1:13     1:00       1    vt01
```

Zhores Queue (partition) structure

- The queue (partition) composition on Zhores:
  - Special partitions:
    - QUEUES: res (gn[21-25]), chess (cn[01-16], gn[16-20], ct[01-04]), gpu_a100 (gn26)
  - Available to all users:

No sharing of resources

Characteristic/Queue	**cpu**	**gpu**	**mem**	**gpu_devel**	**htc**
# nodes per task	unlimited	unlimited	unlimited	2	**1**
Max processing time /default time	6/1 day	6/1day	6/1 day	12/- h	1/- day
Oversubscribe	no	no	no	2	no
QoS	cpu	gpu	mem	gpu_devel	htc
#nodes/CPUs	32/768	15/540	6/480	14/56	83/2852
Nodes	cn[17-44] hd[01-04]	gn[01-15]	ct[05-10]	vt[01-14]	ct[01-10] cn[01-44] hd[01-04] gn[01-25]

- Need to specify the parameters to fully allocate job resources in a queue:
  - To submit into the cpu partition a job expected to run 45 minutes on 4 nodes using 16 cores each and 100 GB memory used per node:

```
sbatch --partition=cpu --nodes=4 –ntasks=16  --mem=100G --time=45:00 runscript.sh
```

  - Same can be done within the script:
    - Ex.: `sbatch runscript.sh`

```
#SBATCH –p cpu
#SBATCH –N 1
#SBATCH –n 16
#SBATCH –mem=100G
#SBATCH –t 45:00
```

SLURM - LAB

- Admin commands to see configuration, queues, etc:
  - Composition of queues
    - sinfo
    - scontrol show partitions
    - squeue (as root to see all jobs)
- To allocate resources for running jobs (semi-)interactively
  - salloc –p gpu_devel –t 5 –N 1 –n 1
  - srun [–w vt11] x.sh
  - srun ls
  - srun hostname
  - squeue [-j 1169093]
  - scontrol show job 1169093
  - scontrol show nodes vt11

Uses previously allocated resources

tasks

```
#!/bin/bash
hostname
date
sleep 45
```

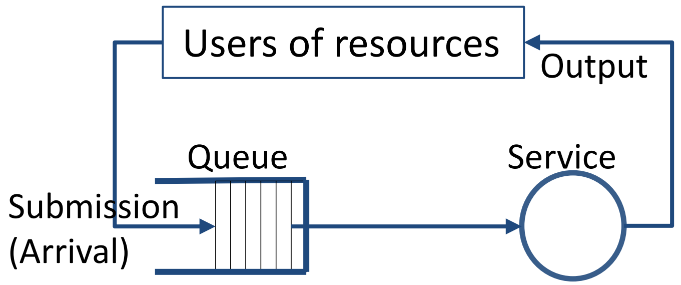
```
bash-4.2$ salloc -p gpu_devel -t 5 -N 1 -n 1
salloc: Pending job allocation 1169093
salloc: job 1169093 queued and waiting for resources
salloc: job 1169093 has been allocated resources
salloc: Granted job allocation 1169093
salloc: Waiting for resource configuration
salloc: Nodes vt11 are ready for job
```

```
bash-4.2$ sbatch -p gpu_devel -t 1 -N 1 -n 1 x.sh
Submitted batch job 1169095
bash-4.2$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1169095 gpu_devel     x.sh i.zachar  R       0:02      1 vt11
Bash-4.2$ ls
-rwxrwxr-x 1 i.zacharov i.zacharov    44 Oct 24 20:00 x.sh
-rw-rw-r-- 1 i.zacharov i.zacharov    41 Oct 24 20:01 slurm-1169095.out
```

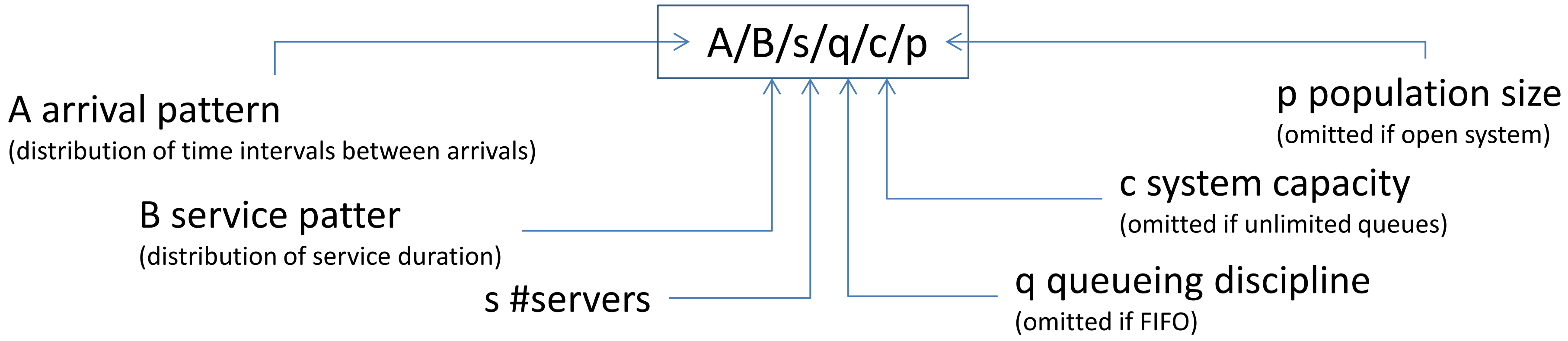
Output file

Theory: Elements of a Queueing System

- Users compete for the resources (services) by generating jobs (requests)
  - Potentially unlimited #jobs (open system), but in practice some bound number (closed system)
- Arrival of jobs in the queue at time t
  - Interval between adjacent arrivals is randomly distributed
- Queue represents jobs waiting for service
  - It can be empty
  - Job in service is not in the queue
  - <u>Queue maximum size</u>: maximum number jobs waiting = System capacity
  - Queueing discipline: organization of inserting to/from queue
    - FIFO = First In First Out (also FCFS = First Come First Serve) – orderly queue
    - LIFO = Last In First Out (also LCFS = Last Come First Serve) - stack
    - SIRO = Serve In Random Order
    - Priority Queue – typically number of queues with different priorities
    - Complex scheduling discipline where jobs change their position based on time spent in q, availability of the service (i.e. “backfill”), etc.
- Service represents activity that takes (processing) time
  - For each job it may take different time (random)
- Queuing Theory gives system performance as function of input parameters
  - Average waiting time as function of the average service time & #arrivals
  - Throughput (#jobs completed per time) as function of average service time & #arrivals



Kendall Classification of Queueing Systems



- A and B can be a
  - Poisson (Markovian time) distribution : M
  - Erlang distrubution: E
  - Deterministic arrivals and/or constant service duration
  - General (any) distribution
- Zhores queueing system is *M/M/n* , where *n* is the number of servers in a queue
  - Zhores has several queues, the analysis if for each queue separately
  - Arrival rate λ $[\frac{1}{s}]$, service rate μ $[\frac{1}{s}]$, **utilization** $\rho = \frac{\lambda}{n\mu} < 1$ (if >1 the queue will grow no bound)
  - Probability for job to join the queue (all n servers occupied)

$$\boxed{C(n, \lambda/\mu) = \frac{1}{1 + (1-\rho)\left(\frac{n!}{(n\rho)^n}\right) \sum_{k=0}^{n-1} \frac{(n\rho)^k}{k!}}}$$

  - Average #jobs in the system:

$$\boxed{N_s = \frac{\rho}{1-\rho} C(n, \lambda/\mu) + n\rho}$$

  - Average response time (waiting time in the system):

$$\boxed{t_w = \frac{C(n, \lambda/\mu)}{n\mu - \lambda} + \frac{1}{\mu}}$$

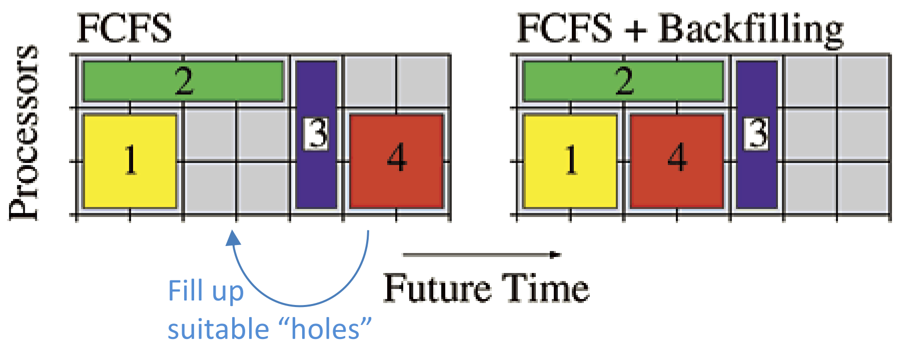
Queueing Theory: conclusions for Zhores

- Estimate resources for your job for best turnaround
  - Specify in sbatch command or job script:
    - Partition, Runtime, #nodes, memory needed, optimal CPU/GPU or task number
- Fastest queue is the queue with smaller average processing time
  - Waiting time is ~1/µ
  - -> Prefer the htc queue for faster turnaround
  - Expect that there could be many jobs in the queue (~100s of jobs)

$$\boxed{t_w \sim \frac{1}{n\mu - \lambda} + \frac{1}{\mu}}$$

- Specify expected Runtime accurately
  - The scheduler *Backfill strategy* may allow you to get results faster
    - Schedule jobs as long as they don't delay a waiting job that is higher in the queue
    - Increases utilization of the cluster (admins will like you)

$$\boxed{N_s \sim \frac{\rho}{1-\rho} + n\rho, \qquad while\ \rho \approx 1}$$



Slurm Interactive jobs - Lab

- Access to an allocated node interactively
  - Eg. for testing specialized (parallel) software that cannot be done on the access node
  - Performance testing: the allocated resources will not be shared

```
srun --time=1:00:00 -p gpu_devel -N 1 -n 1 --pty bash -l
```

    - srun: job 1169972 queued and waiting for resources
    - srun: job 1169972 has been allocated resources
  - Checking: squeue –l

```
JOBID     PARTITION       NAME      USER     STATE            TIME  TIME_LIMI   NODES  NODELIST(REASON)
1169972   gpu_devel       bash     i.zachar  COMPLETING       1:13     1:00        1   vt01
```

- Difference between salloc + srun and standalone srun:
  - The salloc/sbatch allocate resources
    - The srun inherits these resources and executes commands as job steps
    - Multiple srun within a single allocation is possible (multiple job steps)
    - Check job steps with: **`salloc -p htc –N 1 –n 1 –c 4 –t 5 –mem=1G`**
      `./piloop.sh`
      `sacct –j $SLURM_JOB_ID`
    - (this works on Zhores and doesn't work in the sandbox)
    - **This is sometimes called a "Packed Job"**
  - The srun standalone allocates resources and performs one job step
  - Both, salloc and srun block until resources are available (interactive)
  - The sbatch queues the job waiting for the resource allocation

```
#piloop.sh:
# salloc –N 1 –n 1
for ((i=1; i<8; i++))
do
    srun runPlomp 10 &
done
```

SLURM parallel jobs

- Models for Parallel job = tasks running simultaneously
  - A multi-process program
    - Single Process, Multiple Data (SPMD) paradigm, eg. with MPI
  - A multi-threaded program
    - Shared memory paradigm, eg. with OpenMP or pthreads
  - Several instances of a single-threaded program
    - Embarrassingly parallel paradigm, implemented as a job array
  - One master program controlling several slave programs
    - Master/Slave paradigm
- SLURM
  - A task represents a process
    - Request with –ntasks
    - Can be split on several compute nodes
  - A multi-process program is made of several tasks
  - A multi-threaded program is composed on only one task
    - Uses several CPUs, request with –cpus-per-task (or –c in short)
    - Cannot be split across several compute nodes

SLURM OpenMP example

- OpenMP program: make piomp
- Bash script: runomp.sh
  - Start with: sbatch runomp.sh
    - Or: sbatch –c X runomp.sh
  - Most important parameter: -c setting number of cpus per task
  - Note: -N 1 (--nodes)

```
#!/usr/bin/bash
#SBATCH --job-name=piomp
#SBATCH --output=piomp_%j.out
#SBATCH --partition=cpu         # -p cpu
#SBATCH --nodes=1               # -N 1 number of nodes
#SBATCH --cpus-per-task=8       # -c X cpus assigned to each task
#SBATCH --time=2:00
#SBATCH --mem=1G                # memory per node

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}

module load mpi/openmpi-3.1.2
module load compilers/gcc-7.3.1
make piomp &>/dev/null

mtrials=${1:-1}
numtrials=$mtrials
seed=$RANDOM

/usr/bin/time -p ./piomp -t $numtrials -a -t $numtrials -s $seed
```

SLURM MPI example

- MPI program: make pimpi
- Bash script: runmpi.sh
  - Start with: sbatch runmpi.sh
  - Note: -N (--nodes) = -n (ntasks) for distribution of MPI processes to nodes
  - Optimization: leave N unspecified (lump MPI processes on the same nodes)
    - Try –n x –N x/2
  - mpirun “–np auto”
    - from resources

```
#!/usr/bin/bash
#SBATCH --job-name=pimpi
#SBATCH --output=pimpi_%j.out
#SBATCH --partition=cpu         # -p cpu
#SBATCH --nodes=4               # -N 4 number of nodes
#SBATCH --ntasks=4              # -n 4 total number of mpi processes
#SBATCH --cpus-per-task=8       # -c X cpus assigned to each task
#SBATCH --mincpus=9             # to ensure sufficient resources for MPI master
#SBATCH --time=2:00
#SBATCH --mem=1G                # memory per node

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
export MPI_NUM_PROCESS=${SLURM_JOB_NUM_NODES:-4}

module load mpi/openmpi-3.1.2
module load compilers/gcc-7.3.1
make pimpi &>/dev/null

mtrials=${1:-1}
numtrials=$mtrials
seed=$RANDOM

/usr/bin/time -p mpirun ./pimpi -a -t $numtrials -s $seed
```

SLURM Embarrassingly parallel example

- Multiple programs running same code with different parameter(s)
  - Same OpenMP program: make piomp will run with OMP_NUM_THREADS=1
    - Can also be compiled without –fopenmp, only scalar code is needed
- Bash script: runarr.sh
- SBATCH –a 1-N%X
  - X is #running same time
- Output is collected in log file (eg. "tee –a log")

```
#!/usr/bin/bash
#SBATCH --job-name=piarr
#SBATCH --output=piarrJ.out    # can use %j and %A to separate
#SBATCH --partition=htc        # -p
#SBATCH --cpus-per-task=1      # do not need parallelism
#SBATCH --time=2:00
#SBATCH --mem=1G               # memory per node
#SBATCH --array=1-100%20

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}

module load compilers/gcc-7.3.1
make piomp &>/dev/null

mtrials=${1:-1}
numtrials=$mtrials
seed=$(($RANDOM*$SLURM_ARRAY_TASK_ID))

#echo " /usr/bin/time -p ./piomp -t $numtrials"
./piomp -a -t $numtrials -s $seed | tee -a log
```

- Analyze the log to assemble the result
  - Ex.: analyze.sh
    - Awk is used in this case
    - Python could be used instead