SLURM Benutzungsanleitung

Der SLURM Workload Manager

SLURM (**S**imple **L**inux **U**tility for **R**esource **M**anagement) ist ein freies Open-Source System zum Management von Batchjobs und Ressourcen, welches es Nutzern des Clusters erlaubt, Aufgaben scriptgesteuert auf dem LUIS Clustersystem ablaufen zu lassen. SLURM ist ein modernes und erweiterbares Batchsystem, welches weltweit von HPC Clustern unterschiedlichster Größe eingesetzt wird. Im Folgenden wird die grundlegende Arbeit mit SLURM auf dem LUIS Clustersystem beschrieben. Detailliertere Informationen finden sich in der offiziellen Dokumentation zu SLURM.

Die folgenden Befehle sind nützlich, um mit SLURM zu interagieren:

- **sbatch** - submit a batch script
- **salloc** - allocate compute resources
- **srun** - allocate compute resources and launch job-steps
- **squeue** - check the status of running and/or pending jobs
- **scancel** - delete jobs from the queue
- **sinfo** - view intormation abount cluster nodes and partitions
- **scontrol** - show detailed information on active and/or recently completed jobs, nodes and partitions
- **sacct** - provide the accounting information on running and completed jobs

Nachfolgend sind einige Beispiele aufgeführt. Weitere Informationen über den jeweiligen Befehl sind der entsprechenden Manpage, z.B. `man squeue`, oder natürlich der offiziellen SLURM Dokumentation zu entnehmen.

Partitionen

In SLURM sind Rechenknoten zu Partitionen gruppiert. Jede Partition kann als separate Warteschlange (“Queue”) angesehen werden, obwohl ein Job in mehrere Partitionen geschickt werden und ein Rechenknoten mehreren Partitionen angehören kann. Jobs bekommen innerhalb einer Partition Ressourcen zugewiesen, um innerhalb einer bestimmten Zeit Aufgaben auf dem Cluster auszuführen. Zusätzlich existiert das Konzept der “job steps” in SLURM. Innerhalb eines Jobs können mit dem Befehl `srun` so genannte “job steps” gleichzeitig oder nacheinander abgearbeitet werden.

Tabelle listet alle Partitionen mit den zugehörigen Parametern auf. Die aufgeführten Limits können durch Nutzer nicht überschritten werden.

Partition name	Max Job Runtime	Max Nodes Per Job	Max Number of Jobs per User	Max CPUs per User	Default Memory per CPU	Shared Node Usage
gpu	24 hours	1	32	no limit	1600 MB	yes

Um die Last auf dem Cluster zu im Rahmen und SLURM reaktionsfähig zu halten, bestehen die folgenden Beschränkungen für die Gesamtzahl an Jobs:

SLURM limits	Max Number of Running Jobs	Max Number of Submitted Jobs
Cluster wide	10000	20000
Per User	64	500

Wenn Sie für Ihre Arbeit angepasste Limits benötigen, schicken Sie bitte eine Email mit kurzer Begründung an `cluster-help@luis.uni-hannover.de`. Je nach den vorhandenen Ressourcen kann es möglich sein, auch spezielle Anforderungen für einige Zeit zu berücksichtigen, ohne die restliche Nutzerschaft über Gebühr zu benachteiligen.

Um die für Sie gültigen Limits aufzulisten, verwenden Sie das Kommando `sacctmgr`. Beispielsweise:

```
sacctmgr -s show user
sacctmgr -s show user format=user,account,maxjobs,maxsubmit,qos
```

Eine Übersicht über alle gegenwärtig verfügbaren Rechenknoten erhalten Sie mit den folgenden Kommandos:

```
sinfo -Nl
scontrol show nodes
```

Die verfügbaren Partitionen und ihre Konfiguration erhalten Sie mit:

```
sinfo -s
scontrol show partitions
```

Interaktive Jobs

Der Rechencluster wird normalerweise und am effizientesten im Batch-Modus verwendet. Interaktive Jobs sind ebenfalls möglich; diese können sinnvoll sein für Dinge wie z.B.:

- working with an interactive terminal or GUI applications like R, iPython, ANSYS, MATLAB, etc.
- software development, debugging, or compiling

Eine interaktive Sitzung auf einem Rechenknoten starten Sie entweder mit `salloc` oder mit `srun`. Im folgenden Beispiel wird ein interaktiver Job abgeschickt, welcher zwei Tasks (das entspricht zwei CPU-Kernen) und 4 GB Arbeitsspeicher für eine Stunde anfordert:

```
[user@login02 ~]$ srun --time=1:00:00 --ntasks=2 --mem-per-cpu=4G --x11 --
pty $SHELL -i
 srun: job 222 queued and waiting for resources
 srun: job 222 has been allocated resources
[user@euklid-n001 ~]$
```

Sobald der Job startet, erhalten Sie eine interaktive Shell (eine “Kommandozeile”) auf dem ersten ihrem Job zugewiesenen Rechenknoten (`euklid-n001` im obigen Beispiel). Die Option `–x11` erstellt eine X11-Umleitung auf diesen ersten Knoten, was die Nutzung grafischer Anwendungen ermöglicht. Die interaktive Sitzung wird beendet, indem die Shell verlassen wird.

Eine interaktive Sitzung mit Zugriff auf GPU Ressourcen muss über das Kommando `salloc` gestartet werden. Das folgende Beispiel belegt für einen Zeitraum von zwei Stunden zwei GPUs pro Knoten:

```
[user@login02 ~]$ salloc --time=2:00:00 --gres=gpu:2
salloc: Granted job allocation 228
salloc: Waiting for resource configuration
salloc: Nodes euklid-n002 are ready for job
```

Sobald eine Reservierung erstellt wurde, startet das Kommando `salloc` eine Shell auf dem **Loginknoten**, auf welchem der Job abgeschickt wurde. Um ihre Anwendung auf den zugewiesenen Rechenknoten zu starten (`euklid-n002` im Beispiel), führen Sie entweder das `srun`-Kommando in der Loginshell aus:

```
[user@login02 ~]$ module load my_module
[user@login02 ~]$ srun ./my_program
```

... oder Sie verbinden sich via ssh mit den von Ihnen belegten Rechenknoten:

```
[user@login02 ~]$ echo $SLURM_NODELIST  # assigned compute node(s)
euklid-n002

[user@login02 ~]$ ssh euklid-n002

[user@euklid-n002 ~]$ module load my_module
[user@euklid-n002 ~]$ ./my_program
```

Um die Sitzung zu beenden, geben Sie `exit` in der Loginshell ein:

```
[user@login02 ~]$ exit
exit
salloc: Relinquishing job allocation 228
salloc: Job allocation 228 has been revoked.
```

Ein Batchscript abschicken

Ein SLURM Jobscript ist nichts anderes als ein Shellscript, welches im Beginn einen Satz spezieller Anweisungen (“Direktiven”) enthält. Direktiven sind durch die Zeichenkette #SBATCH am Zeilenbeginn zu erkennen. Dieses Batchscript wird dann mit dem Kommando `sbatch` an das Batchsystem übergeben.

Beispiel eines seriellen Jobs

Das folgende ist ein Beispiel eines einfachen seriellen Jobscripts (speichern Sie die Zeilen in die Datei `test_serial.sh`).

Hinweis: ändern Sie die #SBATCH-Direktiven so ab, dass sie Ihrem Anwendungsfall entsprechen.

```
#!/bin/bash
#SBATCH --job-name=test_serial
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:20:00
#SBATCH --constraint=[skylake|haswell]
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_serial-job_%j.out
#SBATCH --error test_serial-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load my_module

# Start my serial app
srun ./my_serial_app
```

Um den Batchjob abzuschicken, verwenden Sie

```
sbatch test_serial.sh
```

Hinweis: sobald Rechenknoten für Ihren Job belegt wurden, können Sie sich per `ssh` von den Loginrechnern damit verbinden.

Hinweis: wenn Ihr job versucht, mehr Ressourcen zu verwenden, als Sie mit den #SBATCH-Direktiven angefordert haben, wird er automatisch vom SLURM-Server beendet.

Hinweis: wir empfehlen, in Batchjobs die option `#SBATCH –export=NONE` zu setzen. Ansonsten übergibt SLURM die gegenwärtig gesetzten Umgebungsvariablen an den Job.

Tabelle 1.3 zeigt häufig benutzte sbatch-Optionen, welche entweder im Jobscript über die #SBATCH-Direktive oder auf der Kommandozeile angegeben werden können. Kommandozeilenoptionen überschreiben Optionen im Script. Die Kommandos `srun` und `salloc` akzeptieren die selben Optionen.

sbatch/srun/salloc options. Both long and short options are listed

Options	Default Value	Description
–nodes=<N> or -N <N>	1	Number of compute nodes
–ntasks=<N> or -n <N>	1	Number of tasks to run
–cpus-per-task=<N> or -c <N>	1	Number of CPU cores per task
–ntasks-per-node=<N>	1	Number of tasks per node
–ntasks-per-core=<N>	1	Number of tasks per CPU core
–mem-per-cpu=<mem>	partition dependent	memory per CPU core in MB
–mem=<mem>	partition dependent	memory per node in MB
–gres=gpu:<type>:<N>	-	Request nodes with GPUs

Options	Default Value	Description
–time=<time> or -t <time>	partition dependent	Walltime limit for the job
–partition=<name> or -p <name>	none	Partition to run the job
–constraint=<list> or -C <list>	none	Node-features to request
–job-name=<name> or -J <name>	job script’s name	Name of the job
–output=<path> or -o <path>	slurm-%j.out	Standard output file
–error=<path> or -e <path>	slurm-%j.err	Standard error file
–mail-user=<mail>	your account mail	User’s email address
–mail-type=<mode>	-	Event types for notifications
–exclusive	nodes are shared	Exclusive acccess to node

Eine vollständige Liste der Parameter finden Sie in der sbatch man page: `man sbatch`

Hinweis: ein mit `–mem=0` abgeschickter Job erhält Zugriff auf den vollständigen Speicher der belegten Knoten.

Als Voreinstellung werden die stdout- und stderr-Dateideskriptoren eines Batchjobs in die Dateien `slurm-%j.out` und `slurm-%j.err` umgeleitet. Dabei wird für `%j` die SLURM Batchjob-ID Ihres Jobs eingesetzt. Beide Dateien werden in dem Verzeichnis erstellt, in welchem Sie Ihren Job abgeschickt haben. Mit den Optionen `–output` und `–error` können Sie andere Namen bzw. Verzeichnisse angeben. Die Ausgabedateien werden erstellt, sobald der Job startet, und die Ausgabe wird umgeleitet, während der Job läuft, so dass sie seinen Fortschritt verfolgen können. Da SLURM allerdings die Daten aus Effizienzgründen puffert und nur gelegentlich wegschreibt, werden Sie die Augaben Ihres Jobs nicht unmittelbar sehen. Um dieses Verhalten zu ändern, können Sie entweder im Script oder auf der Kommandozeile `-u` oder `–unbuffered` verwenden. **Wir empfehlen, das NICHT allgemein zu tun, besonders dann nicht, wenn der Job große Datenmengen ausgibt**.

Falls die Option `–error` nicht angegeben wird, werden stdout und stderr in die Datei umgeleitet, welche durch `–output` bestimmt wurde.

Beispiel eines OpenMP jobs

Für OpenMP jobs müssen Sie `–cpus-per-task` auf einen Wert größer eins setzen und zusätzlich die Variable `OMP_NUM_THREADS` definieren. Das Beispielscript startet acht Threads mit **jeweils** 2 GiB Speicher und einer maximalen Laufzeit von 30 Minuten.

```
#!/bin/bash
#SBATCH --job-name=test_openmp
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:30:00
#SBATCH --constraint=[skylake|haswell]
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_openmp-job_%j.out
#SBATCH --error test_openmp-job_%j.err

# Change to my work dir
```

```
cd $SLURM_SUBMIT_DIR

# Bind your OpenMP threads
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export KMP_AFFINITY=verbose,granularity=core,compact,1
export KMP_STACKSIZE=64m

## Load modules
module load my_module

# Start my application
srun ./my_openmp_app
```

Das `srun`-Kommando im Skript oben startet die Umgebung für eine Anwendung auf mehreren CPU-Kernen, aber auf **einem** Knoten. Für MPI Jobs werden Sie möglicherweise mehrere CPU-Kerne auf **mehreren** Knoten verwenden. Dazu sehen Sie sich das folgende Beispiel für einen MPI-Job an.

Hinweis: Statt der “traditionell üblichen” MPI-Startprogramme wie `mpirun` oder `mpiexec` sollte immer `srun` Verwendung finden.

Beispiel eines MPI Jobs

In diesem Beispiel werden auf dem Lena-Cluster 10 Rechenknoten mit jeweils 16 CPU-Kernen und **insgesamt** 320 GiB Speicher für einen Zeitraum von maximal 2 Stunden angefordert.

```
#!/bin/bash
#SBATCH --job-name=test_mpi
#SBATCH --partition=lena
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=16
#SBATCH --mem-per-cpu=2G
#SBATCH --time=02:00:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_mpi-job_%j.out
#SBATCH --error test_mpi-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load foss/2018b

# Start my MPI application
srun --cpu_bind=cores --distribution=block:cyclic ./my_mpi_app
```

Wie oben erwähnt sollten Sie statt `mpirun` oder `mpiexec` besser `srun` verwenden, um Ihre parallele Anwendung zu starten.

Innerhalb des selben MPI-Jobs können Sie `srun` benutzen, um mehrere Anwendungen gleichzeitig zu

starten, wobei jede nur einen Teil der reservierten Ressourcen benutzt. Die bevorzugte Methoden ist es allerdings, ein Job Array zu verwenden (siehe Abschnitt). Das folgende Beispielscript startet gleichzeitig 3 MPI Programme, die jeweils 64 Tasks verwenden (4 Knoten, jeweils 16 Cores), insgesamt also 192 Tasks:

```
#!/bin/bash
#SBATCH --job-name=test_mpi
#SBATCH --partition=lena
#SBATCH --nodes=12
#SBATCH --ntasks-per-node=16
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:02:00
#SBATCH --constraint=[skylake|haswell]
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_mpi-job_%j.out
#SBATCH --error test_mpi-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load foss/2018b

# Start my MPI application
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-node=16
./my_mpi_app_1 &
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-node=16
./my_mpi_app_1 &
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-node=16
./my_mpi_app_2 &
wait
```

Beachten Sie das `wait`-Kommando im Script; dieses bewirkt, dass das Script so lange wartet, bis alle zuvor mit $&$ (Ausführung im Hintergrund) abgeschickten Kommandos beendet sind. Wir bitten höflich darum darauf zu achten, dass die für die Ausführung jedes Einzelkommandos benötigte Zeit nicht zu unterschiedlich zu den anderen ist, damit nicht zu viel wertvolle CPU-Zeit verschwendet wird.

Jobarrays

Jobarrays können verwendet werden, um mehrere Jobs mit den selben Ressourcenanforderungen abzuschicken. Um ein Jobarray zu verwenden, benutzen Sie die Direktive #SBATCH `–array` im Jobscript bzw. die Option `–array` oder `-a` auf der Kommandozeile von `sbatch`. Das folgende Script erstellt 12 Jobs mit den Arrayindizes 1 bis 10, 15 und 18:

```
#!/bin/bash
#SBATCH --job-name=test_job_array
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2G
```

```
#SBATCH --time=00:20:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --array=1-10,15,18
#SBATCH --output test_array-job_%A_%a.out
#SBATCH --error test_array-job_%A_%a.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load my_module

# Start my app
srun ./my_app $SLURM_ARRAY_TASK_ID
```

Innerhalb eines Jobscripts wie im obigen Beispiel können Arrayindizes über die Variable `$SLURM_ARRAY_TASK_ID` abgefragt werden. Die Variable `$SLURM_ARRAY_JOB_ID` hingegen enthält die Job ID des gesamten Job-Arrays. Falls Sie die Anzahl maximal gleichzeitig laufender Jobs innerhalb eines Arrays begrenzen wollen, – z.B. wegen umfangreicher Datentransfers zum/vom dem BIGWORK Dateisystem – können Sie das mit dem %-Trennzeichen tun. Beispiel: die Direktive `#SBATCH –array 1-50%5` erzeugt 50 Jobs, wovon nur jeweils fünf zu jeder Zeit gleichzeitig aktiv sind.

Hinweis: Die maximale Anzahl der Jobs in einem Job-Array ist auf 100 begrenzt.

Umgebungsvariablen von SLURM

SLURM setzt etliche Umgebungsvariablen für die unter seiner Kontrolle laufenden Jobs. Tabelle 1.4 zeigt häufig verwendete Umgebungsvariablen, die sich als nützlich in Ihren Job Scripts erweisen könnten. Eine vollständige Liste finden Sie untr der Überschrift “OUTPUT ENVIRONMENT VARIABLES” in der man page von sbatch.

SLURM environment variables

$SLURM_JOB_ID	**Job id**
$SLURM_JOB_NUM_NODE	Number of nodes assigned to the job
$SLURM_JOB_NODELIST	List of nodes assigned to the job
$SLURM_NTASKS	Number of tasks in the job
$SLURM_NTASKS_PER_CORE	Number of tasks per allocated CPU
$SLURM_NTASKS_PER_NODE	Number of tasks per assigned node
$SLURM_CPUS_PER_TASK	Number of CPUs per task
$SLURM_CPUS_ON_NODE	Number of CPUs per assigned node
$SLURM_SUBMIT_DIR	Directory the job was submitted from
$SLURM_ARRAY_JOB_ID	Job id for the array
$SLURM_ARRAY_TASK_ID	Job array index value
$SLURM_ARRAY_TASK_COUNT	Number of jobs in a job array
$SLURM_GPUS	Number of GPUs requested

GPU-Jobs auf dem Cluster

Einige der Knoten im LUIS-Cluster sind mit NVIDIA Tesla GPU-Karten ausgestattet.

Derzeit sind 4 Dell-Rechner mit jeweils 2 NVIDIA Tesla V100 für die allgemeine Benutzung in der Partition gpu zugelassen.

Mit diesem Kommando können Sie sich den gegenwärtigen Status aller Knoten in der gpu-Partition und die von ihnen bereit gestellten Ressourcen einschließlich Typ und Anzahl der installierten GPUs anzeigen lassen:

```
 $ sinfo -p gpu -NO
nodelist:15,memory:8,disk:10,cpusstate:15,gres:20,gresused:20
  NODELIST      MEMORY  TMP_DISK  CPUS(A/I/O/T)  GRES
GRES_USED
  euklid-n001   128000  291840    32/8/0/40      gpu:v100:2(S:0-1)
gpu:v100:2(IDX:0-1)
  euklid-n002   128000  291840    16/24/0/40     gpu:v100:2(S:0-1)
gpu:v100:1(IDX:0)
  euklid-n003   128000  291840    0/40/0/40      gpu:v100:2(S:0-1)
gpu:v100:0(IDX:N/A)
  euklid-n004   128000  291840    0/40/0/40      gpu:v100:2(S:0-1)
gpu:v100:0(IDX:N/A)
```

Um eine GPU-Ressource anzufordern, müssen Sie Ihrem Job Script (bzw. auf der Kommandozeile) die Direktive #SBATCH –gres=gpu:<type>:n hinzufügen. Dabei stellt “n” die Anzahl der gewünschten GPUs dar. Die Angabe des Typs der gewünschten GPU (<type>) kann weg gelassen werden. Das folgende Jobscript fordert 2 Tesla V100 GPUs, 8 CPUs in der gpu-Partition und 30 Minutes Laufzeit an:

```
#!/bin/bash
#SBATCH --job-name=test_gpu
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --gres=gpu:v100:2
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_gpu-job_%j.out
#SBATCH --error test_gpu-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load fosscuda/2018b

# Run GPU application
```

```
srun ./my_gpu_app
```

Wenn Sie einen Job in die gpu-Partition schicken, **müssen** Sie die Anzahl der GPUs angeben. Ansonsten wird Ihr Job unmittelbar beim Abschicken zurückgewiesen.

Hinweis: Auf den Tesla V100-Knoten dürfen Sie derzeit nur bis zu 20 CPUs für jede angeforderte GPU anfordern.

Kommandos für Jobsteuerung und -statusabfragen

Dieser Abschnitt gibt einen Überblick über oft zur Steuerung und Überwachung von Jobs verwendete SLURM-Kommandos.

Abfragekommandos

Den Status Ihrer Jobs in der Warteschlange können Sie mit dem folgenden Kommando abfragen:

```
$ squeue
```

oder – falls Sie Arrayjobs verwenden und jeweils ein Element eines Jobarrays pro Zeile anzeigen lassen wollen –

```
$ squeue -a
```

Beachten Sie, dass das Dollarzeichen $ für das Shellprompt steht und kein Bestandteil des eigentlichen Kommandos ist. Sie sollten das Dollarzeichen $ also NICHT selbst eingeben.

Die Ausgabe von `squeue` sollte mehr oder weniger aussehen wie dieses Beispiel:

```
$ squeue
 JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   412       gpu     test username PD       0:00      1 (Resources)
   420       gpu     test username PD       0:00      1 (Priority)
   422       gpu     test username  R      17:45      1 euklid-n001
   431       gpu     test username  R      11:45      1 euklid-n004
   433       gpu     test username  R      12:45      1 euklid-n003
   434       gpu     test username  R       1:08      1 euklid-n002
   436       gpu     test username  R      16:45      1 euklid-n002
```

ST verweist auf den Status Ihres Jobs. `JOBID` wird vom System verwendet, um Ihren Job zu identifizieren. `NODELIST` zeigt die dem Job zugewiesenen Knoten, `NODES` die Anzahl der angeforderten Knoten und – für Jobs im Wartezustand (PD) – einen Grund (`REASON`) für den Zustand. `TIME` zeigt die vom Job verbrauchte Zeit. Typische Jobstati sind: wartet (`PENDING(PD)`), läuft (`RUNNING(R)`), beende/durchgelaufen (`COMPLETING(CG)`), abgebrochen (`CANCELLED(CA)`), gescheitert (`FAILED(F)`) und pausiert (`SUSPENDED(S)`). Eine vollständige Liste der Zustände finden Sie im Abschnitt “JOB STATE CODES” der squeue man page.

Sie können das Ausgabeformat und welche Inhalte angezeigt werden mit der Option `–format` bzw. -

o beeinflussen. Wenn Sie z. B. zusätzlich die Anzahl der CPUs und die angeforderte WallTime anzeigen lassen wollen:

```
 $ squeue --format="%.7i %.9P %.5D %.5C %.2t %.19S %.8M %.10l %R"
  JOBID PARTITION NODES  CPUS TRES_PER_NODE ST MIN_MEMORY     TIME
TIME_LIMIT NODELIST(REASON)
    489       gpu     1    32         gpu:2 PD         2G     0:00
20:00 (Resources)
    488       gpu     1     8         gpu:1 PD         2G     0:00
20:00 (Priority)
    484       gpu     1    40         gpu:2  R         1G    16:45
20:00 euklid-n001
    487       gpu     1    32         gpu:2  R         2G    11:09
20:00 euklid-n004
    486       gpu     1    32         gpu:2  R         2G    12:01
20:00 euklid-n003
    485       gpu     1    16         gpu:2  R         1G    16:06
20:00 euklid-n002
```

Sie können das Ausgabeformat von `squeue` standarmäßig ändern, indem Sie in Ihrer `$HOME/.bashrc`-Datei die Formatangabe der Umgebungsvariable `SQUEUE_FORMAT` zuweisen:

```
 $ echo 'export SQUEUE_FORMAT="%.7i %.9P %.5D %.5C %.13b %.2t %.19S %.8M
%.10l %R"'>> ~/.bashrc
```

Die Option `%.13b` in der Variablenzuweisung an `SQUEUE_FORMAT` oben sorgt für eine Darstellung der Spalte `TRES_PER_NODE` in der Ausgabe von squeue. Darin steht die Anzahl der angeforderten GPUs eines jeden Jobs.

Das folgende Kommando zeigt alle Jobschritte (“job steps”) an für Prozesse, die mit `srun` gestartet wurden:

```
 squeue -s
```

Um die geschätzten Startzeiten anzuzeigen und welche Rechenknoten voraussichtlich Ihren wartenden Jobs zugewiesen werden, verwenden Sie

```
 $ squeue --start
  JOBID PARTITION NAME     USER ST          START_TIME  NODES SCHEDNODES
NODELIST(REASON)
    489       gpu test username PD 2020-03-20T11:50:09      1 euklid-n001
(Resources)
    488       gpu test username PD 2020-03-20T11:50:48      1 euklid-n002
(Priority)
```

Ein Job kann aus unterschiedlichen Gründen auf seine Ausführung warten. Falls es mehrere Gründe für diesen Wartezustand gibt, wird nur jeweils einer angezeigt.

- **Priority** - Job hat noch keine ausreichend hohe Priorität in der Warteschlange
- **Resources** - Job hat ausreichende Priorität, wartet aber darauf, dass Ressourcen verfügbar werden

- **JobHeldUser** - durch Nutzerbefehl am Starten gehindert
- **Dependency** - wartet auf Ende eines anderen Jobs
- **PartitionDown** - Warteschlange nimmt derzeit keine neuen Jobs an

Die vollständige Liste finden Sie im Abschnitt “JOB REASON CODES” der `squeue` man page.

Benötigen Sie detailliertere Informationen über einen Job, verwenden Sie

```
$ scontrol -d show job
```

Einen detaillierten Status eines bestimmten Jobs rufen Sie auf mit

```
$ scontrol -d show job <job-id>
```

Ersetzen Sie `<job-id>` durch die ID ihres Jobs.

Beachten Sie, dass das Kommando `scontrol show job` den Status von Jobs bis zu 5 Minuten nach deren Abschluss anzeigt. Für Batchjobs, welche vor mehr als 5 Minuten beendet wurden, müssen Sie das `sacct`-Kommando verwenden, um die Statusinformationen aus der SLURM-Datenbank abzufragen (siehe Abschnitt).

Das `sstat`-Kommando zeigt für laufende Jobs Statusinformationen in Echtzeit an, z.B. CPU-Zeit, Verwendung virtuellen Speichers (VM), physisch verwendeter Speicher (Resident Set Size, RSS), Ein-/Ausgabeoperationen (Disk I/O) usw.

```
# show all status fields
sstat -j <job-id>

# show selected status fields
sstat --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j <job-id>
```

Hinweis: obige Kommandos zeigen nur Ihre eigenen Jobs in der SLURM-Warteschlange an.

Befehle zur Jobsteuerung

Der folgende Befehl bricht einen Job mit der ID `<job-id>` ab:

```
$ scancel <job-id>
```

Alle Ihre Jobs entfernen Sie aus der Warteschlange durch

```
$ scancel -u $USER
```

Um nur die Array ID `<array_id>` im Job-Array `<job_id>` abzubrechen:

```
$ scancel <job_id>_<array_id>
```

Wenn im obigen Kommando nur die ID des Job-Arrays angegeben wird, werden alle Elemente dieses Job-Arrays abgebrochen.

Die Befehle oben senden den jobs zuerst das Signal SIGTERM, nach weiteren 30 Sekunden – sofern der Job weiterläuft – ein SIGKILL.

Die Option `-s` ermöglicht Ihnen, jedes beliebige Signal an einen laufenden Job zu senden, was bedeutet, dass SIe direkt von der Kommandozeile mit einem Job kommunizieren können, sofern dieser dafür vorbereitet wurde:

```
$ scancel -s <signal> <job-id>
```

Ein wartender Job kann (zurück)gehalten, also von der Jobplanung ausgenommen werden durch

```
$ scontrol hold <job-id>
```

Um einen zuvor (zurück)gehaltenen Job wieder freizugeben, verwenden Sie

```
$ scontrol release <job-id>
```

Nach Übergabe eines Jobs an das System und während er noch auf die Ausführung wartet, können viele der Parameter noch verändert werden. Typische Daten, die modifiziert werden können sind z.B. Jobgröße (Speichermenge, Anzahl der Rechenknoten, Rechenkerne, Tasks, GPUs), Partition, Abhängigkeiten und Laufzeitgrenzen. Hier sind einige Beispiele:

```
# modify time limit
scontrol update JobId=279 TimeLimit=12:0:0

# change number of tasks
scontrol update jobid=279 NumTasks=80

# change node number
scontrol update JobId=279 NumNodes=2

# change the number of GPUs per node
scontrol update JobId=279 Gres=gpus:2

# change memory per allocated CPU
scontrol update Jobid=279 MinMemoryCPU=4G

# change the number of simultaneously running jobs of array job 280
scontrol update ArrayTaskThrottle=8 JobId=280
```

Eine vollständige Liste der änderbaren Auftragsparameter entnehmen Sie Abschnitt “SPECIFICATIONS FOR UPDATE COMMAND, JOBS” der `scontrol` man page.

Job Accounting-Befehle

Das `sacct`-Kommando zeigt an, welche “Abrechnungsdaten” für aktive und ausgeführte Aufträge in der SLURM-Datenbank gespeichert wurden. Hier sind einige Beispiele:

```
#  list IDs of all your jobs since January 2019
```

```
sacct -S 2019-01-01 -o jobid

# show brief accounting data of the job with <job-id>
sacct -j <job-id>

# display all job accountig fields
sacct -j <job-id> -o ALL
```

Die vollständige Liste der Abrechnungsdatenfelder finden Sie in Abschnitt “Job Accounting Fields” der `sacct` man page. Sie können ebenso das folgende Kommando verwenden: `sacct –helpformat`