Einführung in LiDO3

Bei Fragen / Anregungen bitte E-Mail an: lido-team.itmc@lists.tu-dortmund.de

Ingo Schulz

Fakultät für Informatik / ITMC CC:HPC

TU Dortmund

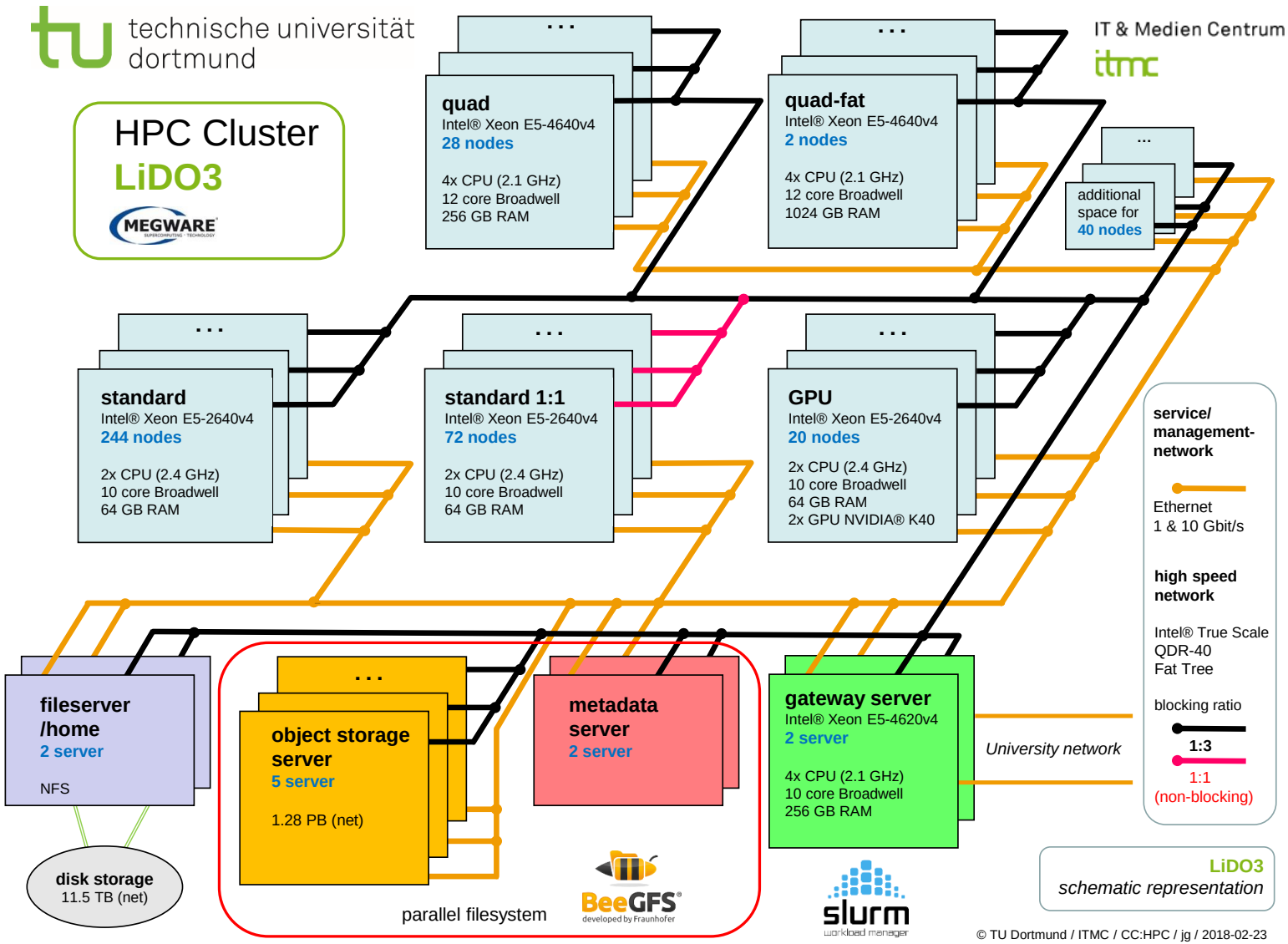
Inhalt der Einführung

- Allgemeines und technische Daten
- Zugang zu LiDO3
- Das Dateisystem
- Scheduling mit Slurm
- Das Modulsystem

LiDO3 – Allgemeines

- LiDO3 = Linux Cluster Dortmund (3. Generation)
- Inbetriebnahme 2018
- Heterogenes Cluster bestehend aus 366 Rechenknoten (8160 CPU Cores)
- Mehr als 30TB RAM verteilt auf die Knoten, pro Knoten mind. 64GB RAM
- Paralleles Dateisystem BeeGFS mit 1,28 PB (sichtbar von jedem Knoten)
- Pro Knoten jeweils mindestens 2TB lokaler Plattenspeicher
- Netzwerkanbindung Infiniband QDR (40 Gbit/s) mit höchstens 1:3 Blocking
- Warteschlangen und Schedule System Slurm
- 2 Gatewayserver
- Software: Compiler (GCC, Intel, PGI...), Profiler/debugger (Allinea), Anwendungen (Matlab, CFX, LS-Dyna, Gaussian, R ....)

Einführung LiDO3



Knotenarten (öffentlich für alle LiDO3 Nutzer)

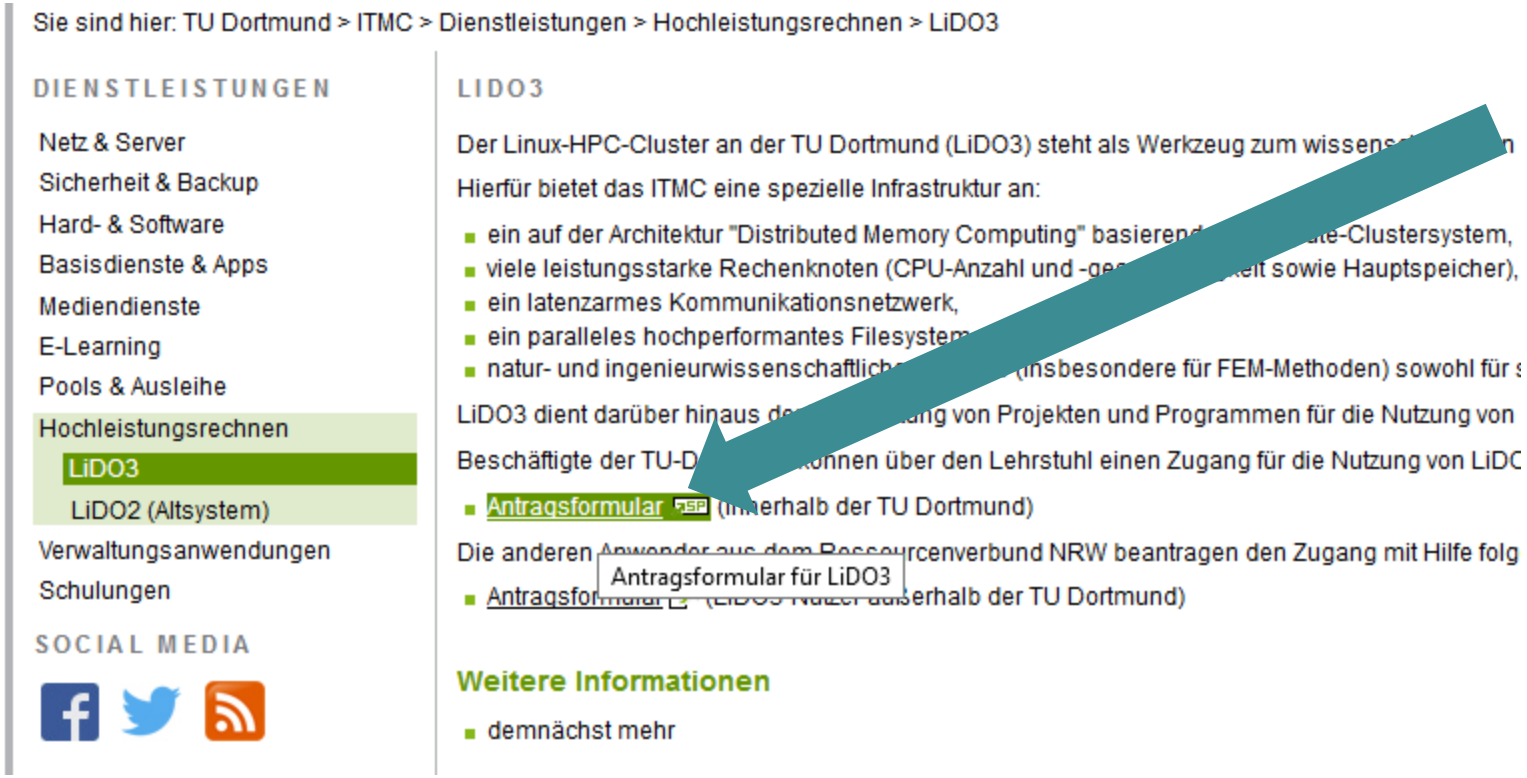
- 2 Sockel Knoten (316 Knoten)
  - Intel Xeon E5-2640v4 – 10 Kerne
  - Taktfrequenz 2,4 GHz
  - L3 Cache 25 MB
  - RAM 64 GB
  - Interconnect Infiniband QDR (davon 72 Knoten mit 1:1 Blocking)
- 4 Sockel Knoten (30 Knoten)
  - Intel Xeon E5-4640v4 – 12 Kerne
  - Taktfrequenz 2,1 GHz
  - L3 Cache 30 MB
  - RAM 256 GB (Ausnahme davon: 2 Knoten mit je 1 TB RAM)
  - Interconnect Infiniband QDR

GPU-Knoten

- 2 Sockel Knoten (20 Stück)
  - Intel Xeon E5-2640v4 – 10 Kerne
  - Taktfrequenz 2.4 GHz
  - L3 Cache 25 MB
  - RAM 64 GB
  - Interconnect Infiniband QDR
  - 2x GPU Beschleuniger Nvidia Tesla K40

Beantragung der LiDO3 Nutzung

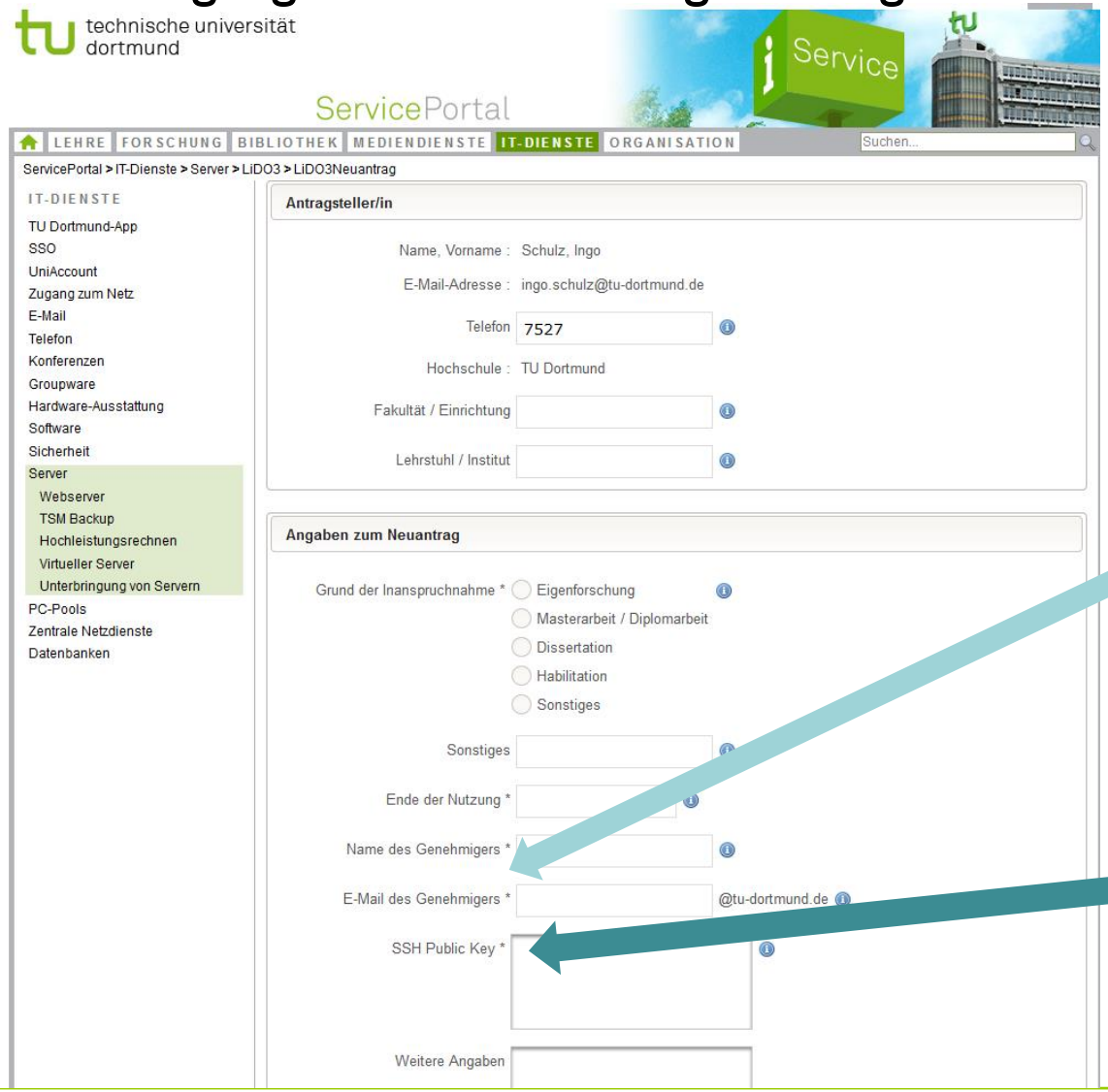
- Für nicht-TU Dortmund Angehörige:
  Antrag mit Formular an das Service-Desk des ITMC der TU Dortmund



Beantragung der LiDO3 Nutzung

- Für Angehörige der TU Dortmund: Antrag über das Serviceportal

Beantragung der LiDO3 Nutzung – Antragsformular



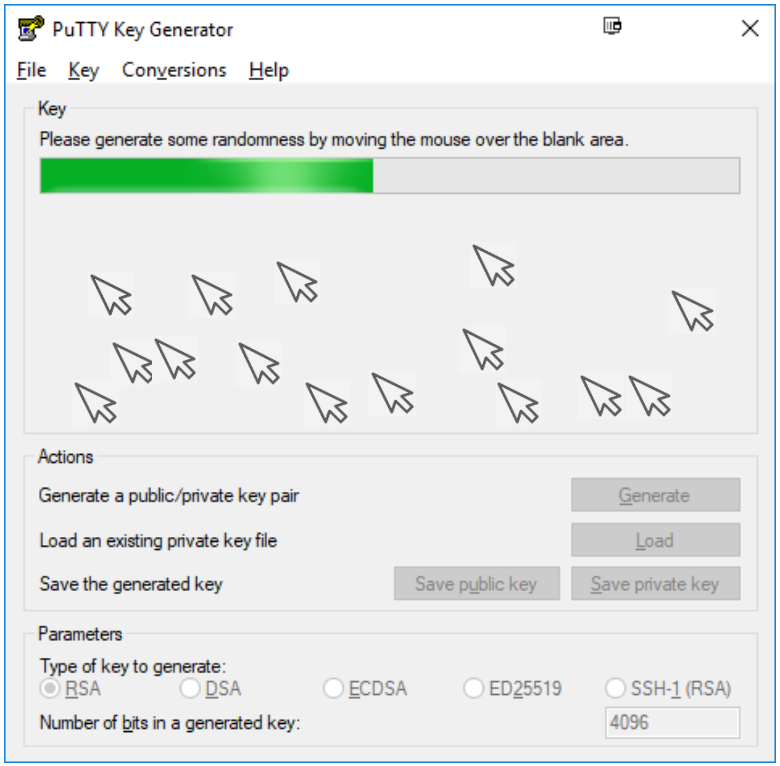
- Ein SSH Public Key ist für den Login obligatorisch
- Username wird der Uni Account in der Form sm... oder m... sein
- Genehmiger ist in der Regel ein(e) Hochschullehrer/-in oder Institutsleiter/-in. Dem Genehmiger wird der Antrag per E-Mail vorgelegt.
- Die Erzeugung und Nutzung des Public Keys wird auf den nachfolgenden Seiten erläutert

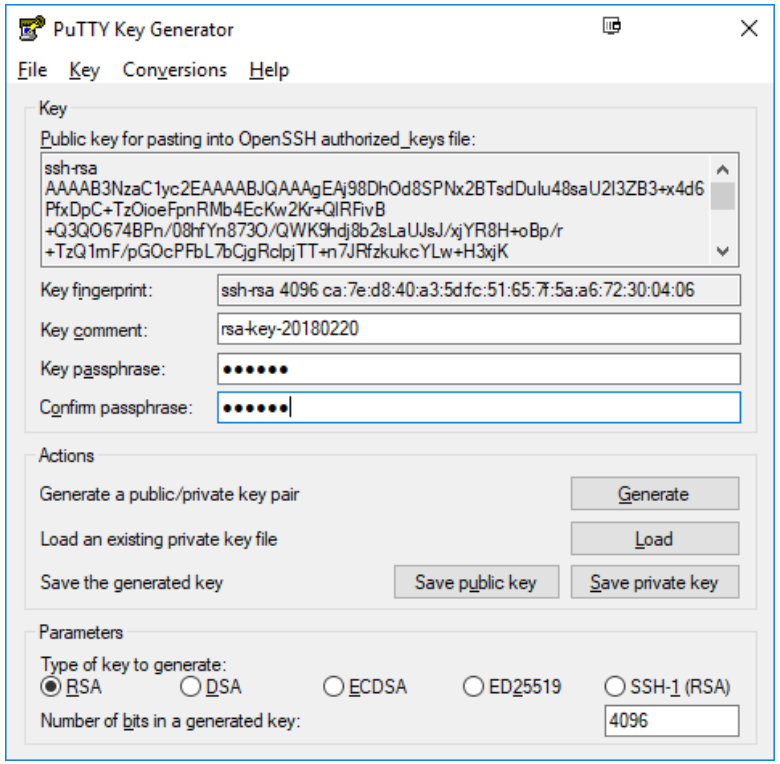
Login (nur mit SSH Public Key!)

- erfolgt per SSH Protokoll ausschließlich auf den Gateways
  - gw01.lido.tu-dortmund.de und
  - gw02.lido.tu-dortmund.de
- unter Linux den ssh Befehl nutzen, z.B. `ssh -i privkey smnutzer@gw....`
- unter Windows einen ssh Clienten wie z.B. „Putty“ nutzen
  - Download unter http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Empfehlenswert: Profil bei Putty einrichten
  - Unter Session einen der o.g. Gateways eintragen und SSH wählen
  - Unter Connection->Data den LiDO3 Usernamen (Uni Account) eingeben
  - Nicht vergessen: Speichern des Profils unter Session
    - Name bei „Saved Sessions“ vergeben und *Save* klicken
  - Mit ggfs. *Load* und *Open* wird die Sitzung auf LiDO3 gestartet.

LiDO3 – Zugang mit Zertifikaten

- LiDO3 Anmeldung erfolgt ausschließlich über Zertifikate (SSH Public Key)
- Windows: Erzeugen der Zertifikate mit puttygen





- Passwort setzen und privaten Schlüssel speichern
- Inhalt des Public Key Fensters kopieren und beim LiDO3 Antragsformular im Serviceportal in das Feld „SSH Public Key“ eingeben!

LiDO3 – Zugang mit Zertifikaten

- Anmeldung mit Hilfe des Agenten „pageant“
  - Startmuster: `pageant c:\speicherort\zertifikat.ppk`
  - Einmalig das Passwort des Zertifikates eingeben
  - Es erscheint das Pageant Symbol in der Taskleiste.

  - Mit rechter Maustaste Kontextmenü öffnen und das gespeicherte Profil wählen.
- Linux: Erzeugung und Nutzung des öffentlichen und privaten Schlüssels
  - In das .ssh Verzeichnis wechseln `>cd ~/.ssh`
  - `>ssh-keygen -t rsa -b 4096`
  - Danach gewünschten Name des Schlüssels und Passwort eingeben
  - Öffentlichen Schlüssel beim LiDO3 Antrag im Serviceportal in das Feld „SSH Public Key“ eingeben!
  - Beim erstmaligen Login wird Schlüsselpasswort abgefragt.
    - Empfehlenswert: Nutzung eines Schlüsselbundes

LiDO3 – Dateisystem

Das zentrale Home Dateisystem ZFS

- LiDO3 verfügt über 14TB „Festplattenspeicher“ für Home Verzeichnisse
- Jeder Nutzer hat ein home Verzeichnis: /home/<username>
  - Limit: 100000 Dateien und 32 GB Datenvolumen insgesamt
  - Daten in diesem Ordner werden regelmäßig gesichert.
- ACHTUNG!!! **Die Homeordner sind auf den Rechenknoten nur read-only gemountet!!!**

Das zentrale parallele Dateisystem BeeGFS

- LiDO3 verfügt über 1280TB „Festplattenspeicher“ für Work Verzeichnisse
  - jeder Nutzer verfügt über das Verzeichnis /work/<username>
  - Daten werden nicht gesichert, d.h. vollständiger Datenverlust möglich
  - Nutzer ist selbst verantwortlich für Datensicherung
  - Ausfall und damit verbundener Datenverlust ist schon vorgekommen!!

Der Scratch Ordner

- Extrem viele und zeitintensive Dateioperationen mit besonders vielen kleinen Dateien belastet BeeGFS
- Abhilfe: Nutzung des lokalen Ordners `/scratch`
- `/scratch` existiert auf jedem Knoten ist aber lokal, d.h. es existiert nur im lokalen Plattenspeicher des Knotens und ist daher nur dort sichtbar
- Lokale Platten haben mindestens 2TB Speicherkapazität
- Für `/scratch` steht fast die ganze Festplatte zur Verfügung (1,8 TB)
- Daher Empfehlung:
  - Bei Jobstart jeweils Daten von `/home` nach `/scratch` kopieren
  - Daten werden jeweils in `/scratch` verarbeitet
  - Bei bzw. vor Jobende Daten wieder zurück von `/scratch` nach `/work` kopieren (nach /home geht nicht, da read-only gemountet!)
  - Wichtige zu sichernde Ergebnisse später von /work nach /home kopieren

Dateien bearbeiten (lokal oder remote)

- Dateien können vorbereitend innerhalb von /work oder von /home nach /work kopiert werden
- Dazu steht Linux Shell Bash (Konsole) zur Verfügung
- Für eine Bash Einführung:
  - http://tldp.org/LDP/Bash-Beginners-Guide/html/
- Programmdateien können zwischen Rechnern kopiert werden
  - Via Linux mit scp
  - Via Windows mit Hilfe einer SCP Software wie z.B.: WinSCP, Filezilla...
- Bearbeitet werden die Dateien mit einem Editor, es gibt auf LiDO3:
  - vi
  - Emacs
  - Gedit
  - Nano

- **Tipp:** Zum kopieren oder anderen Dateioperationen und zum Editieren steht auf den Gatewayrechnern der Midnight Comander zu Verfügung
  Dazu den Befehl `mc` in der Konsole eingeben.
- Natürlich kann man die Dateien auch lokal bearbeiten und mit scp übertragen

Resourcenmanagement

- Jobs werden bei LiDO3 durch das Schedulingsystem Slurm priorisiert
- Warteschlangen werden in Slurm Partionen genannt
- Es gibt verschiedene Partitionen
  - Öffentliche, abhängig von der maximalen Walltime, d.h. der maximal möglichen zur Verfügung stehenden Rechenzeit, wenn der Job startet
  - Private, bestimmter Forschergruppen
- Übersicht der öffentlichen Partitionen

Partition	Max Walltime	Bemerkungen
short	02:00:00	-
med	08:00:00	-
long	48:00:00	-
ultralong	672:00:00	Keine GPU und „non-blocking“ Knoten nutzbar

Resourcenmanagement

- Standardmäßig stehen 512 MB RAM pro Core zur Verfügung
- Bei der Joberstellung können gewünschte Features angegeben werden:
  - Übersicht der wichtigsten Features: (alle CPU Typen sind Intel Xeon)

Feature	CPU Typ	GPU Typ	Max Cores	Max RAM/Knoten
cstd01	E5-2640v4	-	20	64 GB
cstd02 oder ib_1to1	E5-2640v4	-	20	64 GB
cquad01	E5-4640v4	-	48	256 GB
cquad02	E5-4640v4	-	48	1024 GB
tesla_k40	E5-2640v4	2x Nvidia K40	20	64 GB

- Wenn keine Spezialanforderung wie z.B. tesla_k40 benötigt wird, kann auf eine Featureangabe verzichtet werden

Jobmanagement (auf einem der Gateways) via Slurm

- Per Slurm Script (Syntax später)
  - `sbatch meinscript.slurm`
- Interaktiv
  - zunächst Knoten für Job reservieren mit Hilfe von `salloc`:
    - Syntax:
    - `salloc –N` *x* `–C` *F* `–c` *c* `--mem=`*y* `–t` *w* `–p` *part*
    - Dabei werden x Knoten mit Feature *F* angefordert. Man benötigt höchtens *c* Cores, *y* MB pro Knoten an RAM und *w*=hh:mm:ss an Walltime. Die Anforderung wird der Partition *part* zugeordnet.
  - Dann Konsole auf einem der reservierten Knoten erhalten, wenn sie zugewiesen wurden:
    - `srun --pty bash`

Jobmanagement (auf dem Gateway) via Slurm

- Jeder Job (schon in der Wartephase) hat eine Job ID
- Mittels Befehl `squeue -u` *`username`* auf dem Gateway lässt sich die ID sowie zusätzliche Informationen ermitteln
- Löschen eines Jobs aus der Warteschlange bzw. vorzeitiges Beenden mittels `scancel` *`ID`*
- Graphische Übersicht über die Jobs erhält man mit `sview`.
  Dazu ist es erforderlich, dass die ssh Sitzung mit X-Umleitung gestartet wurde. Weiterhin muss auf dem eigenen Rechner ein X-Server aktiv sein.

Slurm Scripts

- Ermöglichen das bequeme automatische Starten eines Jobs

- Verschiedene Zeilen legen die Eigenschaften des Jobs fest, z.B.:
  - Walltime
  - Partition
  - Speicher
  - Prozesse pro Knoten
  - Knoteneigenschaft
  - Email Benachrichtigungen
  - Spezielle Ausgabedatei
  - Programmaufruf

Slurm Script Beispiel

```
#!/bin/bash -l


# The following line asks for usage of 1 compute node with 2
GPUs for 10 minutes using

# 20 cores per node for a non-threaded code. See 'man sbatch'
for details.

#SBATCH --time=10

#SBATCH --nodes=1  --cpus-per-task=20 –-constraint=cgpu01
    --gres=gpu:2

## or: #SBATCH –N 1 –c 1 –C cgpu01 --gres=gpu:2

#SBATCH --partition=long

# Possible 'constraint' values:

#              all, public

#              cgpu01  OR  gpu     OR  tesla_k40
```

Slurm Script Beispiel

```
#                cquad01 OR  xeon_e54640v4
#                cquad02 OR  xeon_e54640v4
#                cstd01  OR  xeon_e52640v4   OR  ib_1to3
#                cstd02  OR  xeon_e52640v4   OR  ib_1to1 OR
nonblocking_comm
# Maximum 'mem' values depending on constraint (values in MB):
#                cstd01/xeon_e52640v4/ib_1to3 AND
#                cstd02/xeon_e52640v4/ib_1to1/nonblocking_comm:
62264
#                cquad01: 255800
#                cquad02: 1029944
#SBATCH --mem=60000
##SBATCH --mem_bind=verbose,local --hint=memory_bound
```

Slurm Script Beispiel

```
#SBATCH --mail-user=test.user@tu-dortmund.de

# Possible 'mail-type' values: NONE, BEGIN, END, FAIL, ALL (=
BEGIN,END,FAIL)

#SBATCH --mail-type=ALL

#SBATCH --signal=15@30
```

Slurm Script Beispiel

```
cd /work/user/workdir


module purge

module load pgi/17.5

export OMP_NUM_THREADS=20

echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
$SLURM_TASK_PID) on $SLURMD_NODENAME"

echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"

echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"



srun ./myapp
```

Auswahl einiger SBATCH Parameter

- `#SBATCH -Nx-y`
  - **Anzahl der Knoten wird auf mindestens x und höchstens y festgelegt. Die Angabe von -y ist optional**
- `#SBATCH -p partitionname`
  - Angabe der Partition
- `#SBATCH -t walltime`
  - Angabe der Walltime in Minuten oder in `d-hh:mm:ss`
- `#SBATCH -J myjobname`
  - Festlegung des Jobnamens
- `#SBATCH -o mypath/filename`
  - Pfad und Dateiname für die Ausgabe (für mögliche Formatoptionen mit Slurmvariablen bitte manpage von `sbatch` aufrufen)

Auswahl einiger SBATCH Parameter

- `#SBATCH -C Constraint`
  - Nur Knoten mit dem Feature `Constraint` werden angefordert
- `#SBATCH --gres=gpu:x`
  - Anfordern von *x* Grafikkarten. (Hinweis: `#SBATCH -C gpu01` reicht allein nicht aus um mit GPUs zu arbeiten!)
- `#SBATCH -c x`
  - *x* Cores pro Task werden angefordert
- `#SBATCH -n y`
  - *y* Tasks werden maximal pro Jobstep ausgeführt
- `#SBATCH --ntasks-per-node=z`
  - *z* Tasks pro Knoten sollen ausgeführt werden. Bei Angabe von `#SBATCH -n y` wird *z* nur als Höchstzahl angesehen.
- `#SBATCH --mem=x`
  - x MB RAM werden pro Knoten benötigt.

Auswahl einiger SBATCH Parameter

- `#SBATCH --mail-user=`*`name@tu-dortmund.de`*
  - Legt die Email Benachrichtigungsadresse fest
- `#SBATCH --mail-type= BEGIN`
  - Mailbenachrichtigung bei Jobstart
- `#SBATCH --mail-type= END`
  - Mailbenachrichtigung bei Jobende
- `#SBATCH --mail-type= FAIL`
  - Mailbenachrichtigung bei Jobabbruch
- `#SBATCH --mail-type= ALL`
  - Mailbenachrichtigung in allen Fällen
- `#SBATCH --export=`*`var1,..,varn`* (Umgebungsvariablen *var1* bis *varn* stehen dem Job zur Verfügung, Alternativ: bei `=ALL` werden alle und bei `=NONE` keine Umgebungsvariable übernommen)

Einführung

- Software, die für Entwicklung und Ausführung benötigt wird, wird in Module organisiert
- Module verändern dynamisch die Umgebung des Nutzers. Sie ermöglichen:
  - Eine saubere Umgebung, d.h. zunächst ist keine Software sichtbar
  - Installation konkurrierender Softwareversionen der gleichen Software
  - Benutzung von Software, die sich gegenseitig ausschließen würde
- Mit `module list` werden aktuell geladene Module angezeigt
- Mit `module avail` erscheint eine Liste verfügbarer Module
- `module add` *`module_name`* lädt das Modul *`module_name`*
- `module rm` *`module_name`* entlädt das entsprechende Modul
- `module purge` entlädt alle geladenen Module
- stets benötigte Module sollten in der .bash_profile Datei im Homeordner festgelegt werden.

Auswahl an Compilern

- Verfügbare Compiler werden bei Bedarf mit Hilfe des module Befehls geladen
  - Übersicht:

Compiler	Module	Commands
GNU Compiler Collection	`module add gcc`	`gcc, g++, gfortran`
Intel Studio XE	`module add intel`	`icc, icpc, ifort`
Portland PGI Compiler	`module add pgi`	`pgcc, pgCC, pgf77, pgf95`

Helloworld

- Beispiel „Helloworld“ (interaktiver Job)
  - Zunächst einen Knoten reservieren
    - `>salloc –N 1 –t 00:05:00 –p short`
  - Konsole erhalten
    - `>srun --pty bash`
  - Dann gcc Modul laden
    - `>module load gcc`
  - Dann kompilieren
    - `>g++ helloworld.cpp  –o helloworld`
  - Dann ausführen
    - `>./helloworld`