Using File Systems Properly

HPC Cafe, 2022-01-18

HPC Services, RRZE / NHR@FAU

Working with data

https://hpc.fau.de/systems-services/systems-documentation-instructions/hpc-storage/

File systems

- File system == directory structure that can store files
- Several file systems can be “mounted” at a compute node
  - Similar to drive letters in Windows (C:, D:, …)
  - Mount points can be anywhere in the root file system

- Available file systems differ in size, redundancy and how they should be used

RRZE file systems overview

Mount point	Access	Purpose	Technology	Backup	Snap-shots	Data lifetime	Quota
`/home/hpc`	`$HOME`	Source, input, important results	NFS on central servers, small	YES	YES @30 min	Account lifetime	50 GB
`/home/vault`	`$HPCVAULT`	Mid-/long-term storage	Central servers	YES	YES @1/day	Account lifetime	500 GB

`/home/woody`
`/home/saturn`
`/home/titan`
	`$WORK`	Short-/mid-term storage, General-purpose	Central NFS server	(NO)	NO	Account lifetime	500 GB

`/lxfs`	`$FASTTMP` (only within meggie)	High performance parallel I/O	Lustre parallel FS via InfiniBand	NO	NO	High watermark	Only inodes
`/???`	`$TMPDIR`	Node-local dir	HDD/SSD/ ramdisk	NO	NO	Job runtime	NO

Caveats:

- $TMPDIR varies significantly in size across clusters (emmy/meggie: 32 GB RAMdisk only), but generally > 1TB
- $TMPDIR is not always job specific

Problem

Main Problem with NFS (and parallel FS)

- In a job, avoid *accessing* large numbers of files
  `$HOME`, `$HPCVAULT`, `$WORK`, `$SATURNHOME`
- Expensive operations on NFS (and also parallel file systems):
  - Access file stats like creation/modification time, permissions…
  - Opening/closing files
- These cause high load on servers
  - This slows down your job and impacts all other users
- Use instead
  - if supported by application: HDF5, file-based databases
  - pack files into an archive (e.g. tar + optional compression) and use node-local SSDs (huge amounts of file opens are no problem there)

Working with Archives and Node-Local SSDs

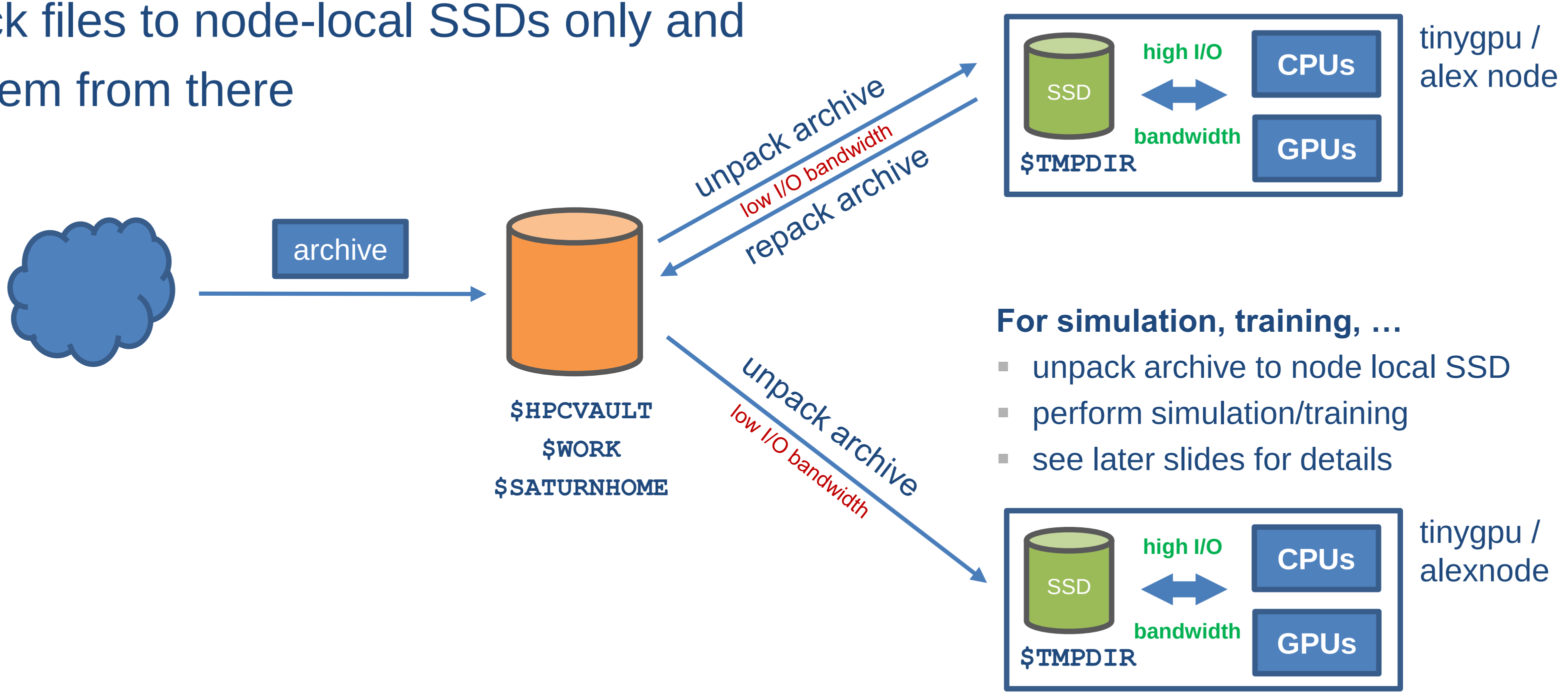
Do not unpack archive to:

`$HOME/$HPCVAULT/$WORK`

Unpack files to node-local SSDs only and use them from there

Optionally: if original archive must be altered

- unpack it to node local SSD (interactive job)
- optionally change files
- repack files and copy back to NFS



For simulation, training, …

- unpack archive to node local SSD
- perform simulation/training
- see later slides for details

Example: Repack Archive with an Interactive Job on tinygpu

```
# request interactive job on tinygpu from woody
$ salloc.tinygpu -t hh:mm:ss --gres=gpu:1

$ WORKDIR="$TMPDIR/$SLURM_JOBID"
$ mkdir "$WORKDIR"
$ cd "$WORKDIR"

# unpack into current directory
$ tar xf $WORK/archive.tar
# process files …

# pack all files from the current directory
# into a new archive on $WORK
$ tar cf $WORK/new-archive.tar *
# clean up
$ cd ; rm -r "$WORKDIR"
```

Here, tar is just used an example, use whatever you see fit best

Unpacking depending on extension:

```
.tar.bz2: tar xjf $WORK/archive.tar.bz2
.tar.gz:  tar xzf $WORK/archive.tar.gz
.tar.xz:  tar xJf $WORK/archive.tar.xz
```

Packing + compression depending on extension:

```
.tar.bz2: tar cjf $WORK/archive.tar.bz2 *
.tar.gz:  tar czf $WORK/archive.tar.gz *
.tar.xz:  tar cJf $WORK/archive.tar.xz *
```

If compression does not save any space, using tar without compression is also an option

Some benchmark data

Using local file systems for vast amounts of files

Benchmark structure

- lz4 uncompress

```
cd $TMPDIR
f=$SATURNHOME/inputfile.tar.lz4
time (lz4 -d $f | tar xf -)
```

- gunzip

```
cd $TMPDIR
f=$SATURNHOME/inputfile.tar.gz
time tar xzf $f
```

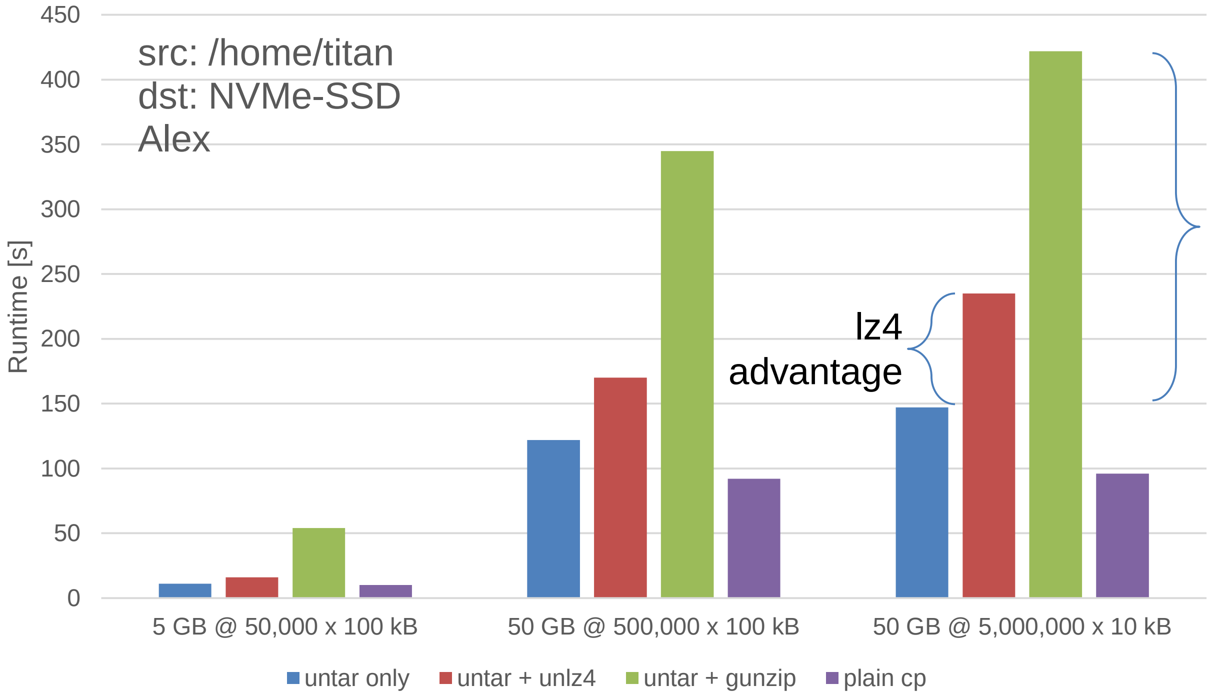
- Plain untar

```
cd $TMPDIR
f=$SATURNHOME/inputfile.tar
time tar xf $f
```

- Data was hardly compressible (random numbers, images)
- Every run was a new job to minimize impact of FS caching

Unpacking to a local disk

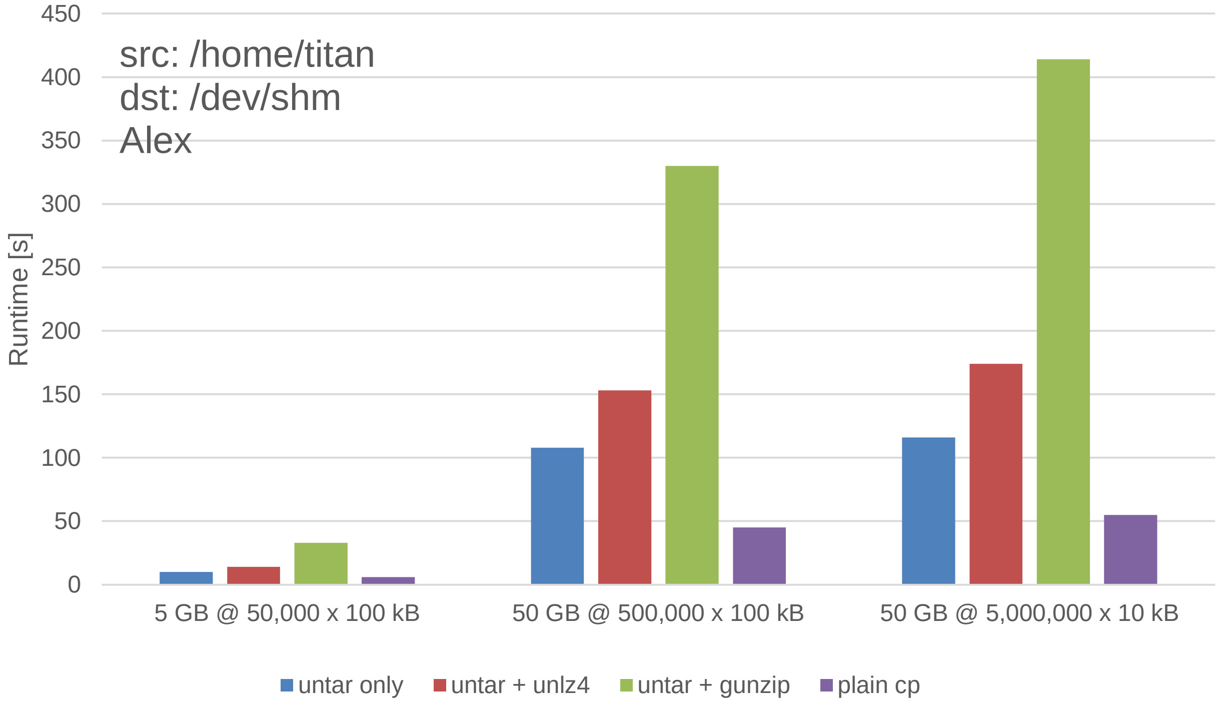
Case 1: NFS → local NVMe-SSD (**`$TMPDIR`**)



Uncompress is major part of runtime

Unpacking to a local disk

Case 2: NFS → local ramdisk (`/dev/shm`)



Caveats:

- /dev/shm is actually system RAM
- Cuts away at your available RAM
- Available space is divided among GPUs in tinyGPU and Alex

Parallel uncompress?

- If compression is effective, it should be used (if data transfer time can be reduced significantly)
- Several solutions
  - Actual concurrent untar/gunzip processes on different archives
  - mpiFileUtils (https://hpc.github.io/mpifileutils/)
  - (un)compress tools with built-in threading
- Poor (wo)man's solution →

```
cd $TMPDIR
f1=${SATURNHOME}/arch1.tar.gz
f2=${SATURNHOME}/arch2.tar.gz

(mkdir 1; cd 1; tar xzf $f1) &
(mkdir 2; cd 2; tar xzf $f2) &
wait
```

Caveats

- Observed FS performance can fluctuate wildly
  - Caching (on server *and* client), server load, network load
- Servers are connected with different wirespeeds
  - 100 GBE vs. 25 GBE vs. 10 GBE
- Servers have different disk technologies (HDD, SSD)

- If (un)compression is required, it may take a long time
  - Consider parallel uncompress (call if you need help)

- Still, the general guidelines are always the same
  - We will support you with benchmarking if required

Some solutions implemented by customers

Many files, frequent accesses

- Training data set with many separate files
- /home/vault
- Many accesses per second to the data set

Remedy

- Load complete data set into RAM at job start

Frequent checkpoints

- Regular checkpoints to /home/woody every 2-5 minutes, 10-200 MB in size
  - Should not be a problem
  - Still, even 5-minute checkpoints are unnecessarily frequent

Frequent metadata accesses

- “Many” files
- Frequent accesses to small files or sections of them

Remedy

- Put files into ZIP/tar archive (better copy performance)
- Unpack to node-local temp directory and work from there
- Cleanup may be automatic

```
$ WORK_DIR=`mktemp.exe -d -p $TMPDIR`
$ cd $WORK_DIR
$ unzip $WOODYHOME/foo.zip
$ # ... Now work with data in $WORK_DIR
$ # Clean up at the end:
$ cd
$ rm -rf $WORK_DIR
```

Frequent metadata accesses

- Many small files on $HOME
- 100-500 kB
- ~ 50 accesses per second

Remedy

- Pre-package files to one HDF5 file
- Load to internal data structure in RAM upon startup

Many files, too large to fit in memory

- Many files in /home/woody
- Frequent reads necessary since whole data set does not fit into RAM
- Repeated accesses to every file

Remedy (a)

- Pack files into ZIP/tar archive, unpack to $TMPDIR at job start

Remedy (b)

- Try to open each file from $TMPDIR, copy from archive if not present (caching)

Questions? Suggestions?