Data storage on Triton: an introduction

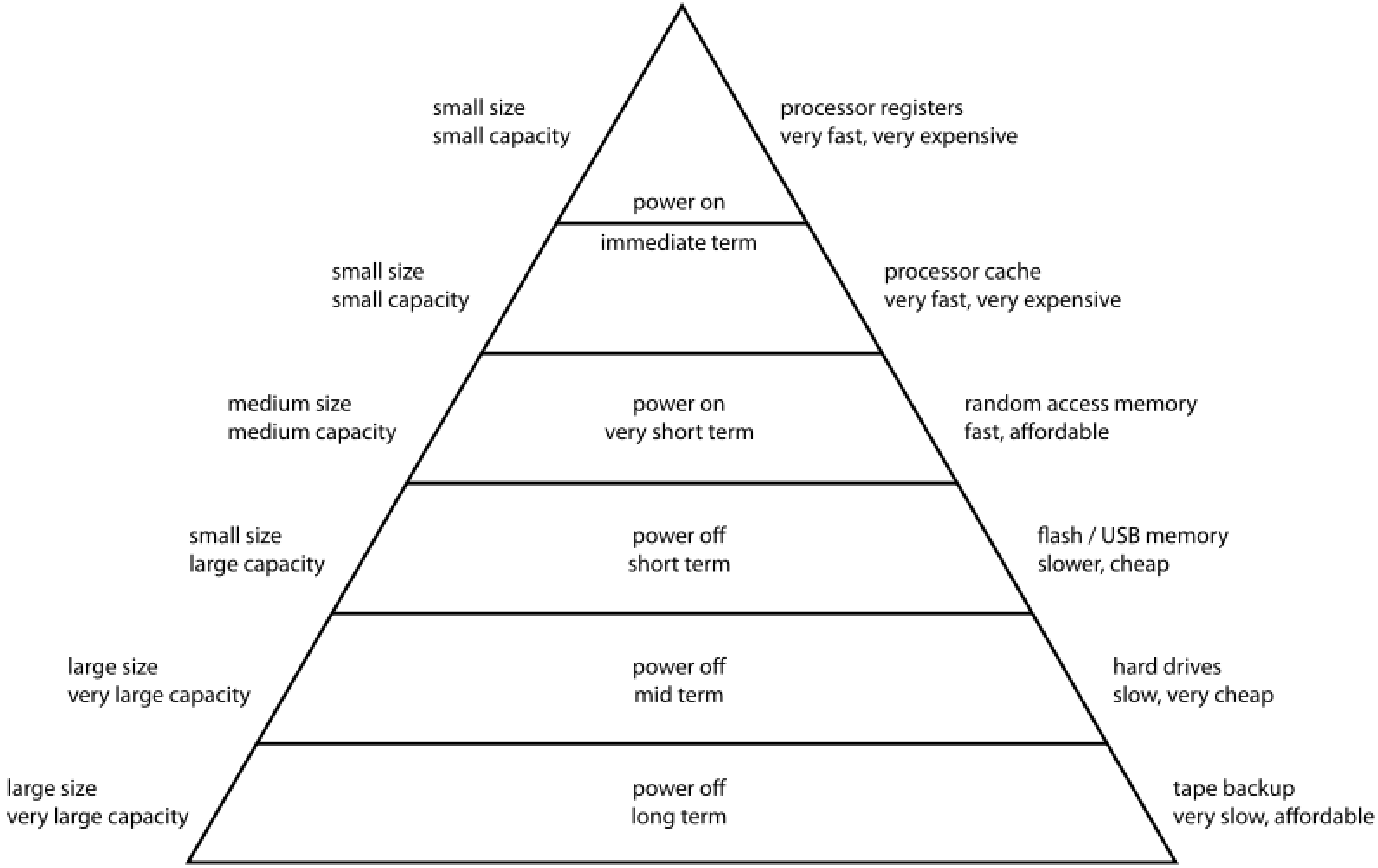
- Motivation
- How storage is organized in Triton
- How to optimize IO
- Do's and Don'ts
- Exercises

Aalto University
School of Science

Data storage: Motivation

- Program speed isn’t just about processor speed: **you have to get data to the processor**
- Dealing with IO properly prevents performance bottlenecks (and this is a **major** factor in computer design)
- Input/output is a shared resource: one user can cause problems to other users
- Your work will be more efficient if you organize your work to suit your data

Computer Memory Hierarchy



small size
small capacity

power on
immediate term

processor registers
very fast, very expensive

small size
small capacity

processor cache
very fast, very expensive

medium size
medium capacity

power on
very short term

random access memory
fast, affordable

small size
large capacity

power off
short term

flash / USB memory
slower, cheap

large size
very large capacity

power off
mid term

hard drives
slow, very cheap

large size
very large capacity

power off
long term

tape backup
very slow, affordable

Network
storage

https://en.wikipedia.org/wiki/Memory_hierarchy

Storage considerations

You have all of these things to think about:

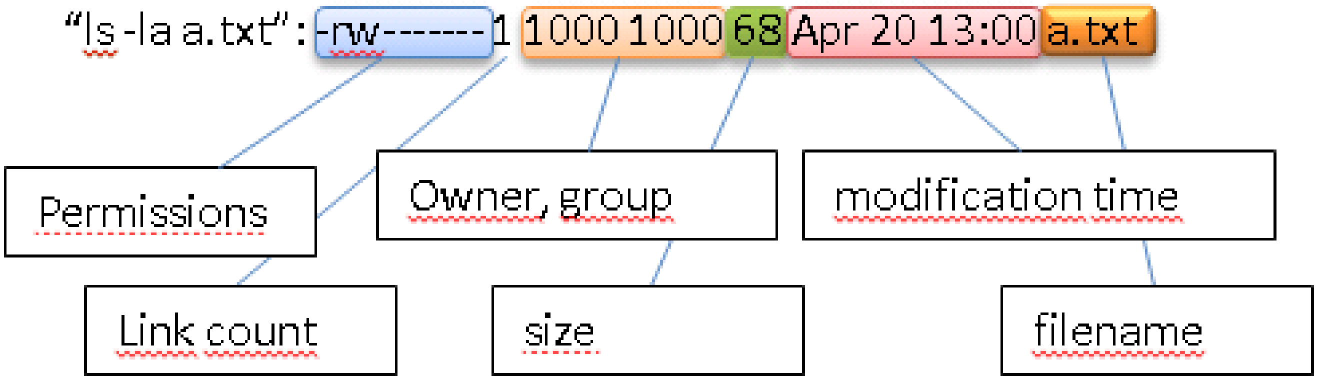
- Network (shared) vs local (dedicated)
- Shared (with a group) vs personal (only you can access)
- Sequential access vs random access (different performances)
- Few large files vs many small files
- Parallel vs single access
- Backed up vs not
- Rotating hard disk vs solid state drive
- You **do** need to put your own effort into using storage properly
  - Using proper file formats and applications
  - Move data to best storage yourself (during calculations)

How storage is organized in Triton

Data stored in files

What is a file?

- File = metadata + contents (block data)
- Accessing contents: **cat a.txt**
- Showing some metadata: **ls -l a.txt**



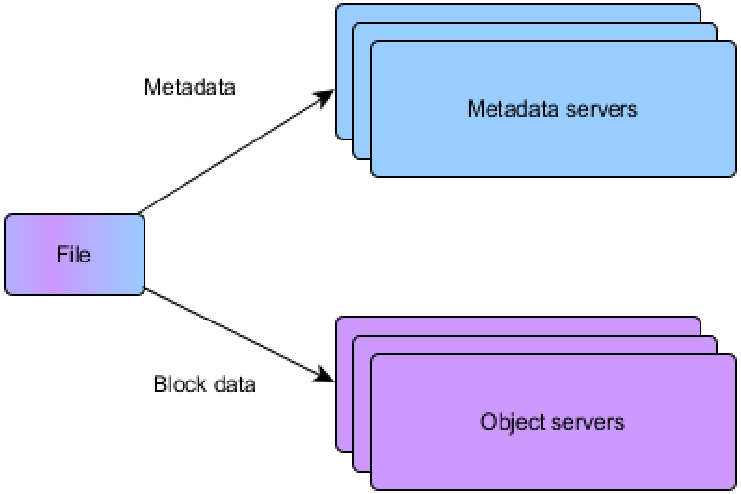
```
"ls -la a.txt": -rw------- 1 1000 1000 68 Apr 20 13:00 a.txt
```

- Full metadata: **stat a.txt**, lsattr, getfattr

Files stored in /scratch

In /scratch, metadata and block data are separated:

- Metadata query answered by metadata server
- Block data query, object storage server answers



File system performance metrics

- Stream I/O and random IOPS
  - Stream measures the speed of reading large sequential data from system
  - IOPS measure random small reads to the system – number of metadata/block data accesses
  - To measure your own application, profiling or internal timers needed
  - Rough estimate can be aquired from /proc/<pid>/io or by using strace

Triton network storage

User's Home folder (NFS filesystem)

- `/home/$username/` -directory
- Shared for every computational node
- Meant for scripts etc. Not highly parallel
- Nightly backup, 1GB quota (small)

Work/Scratch (Lustre filesystem)

- Personal: `/scratch/$department/work/$username/`
- Group: `/scratch/$department/$project/`
- Shared for every computational node
- Meant for fast, highly parallel input/output data but inefficient for small random access
- Quota varies per project, 2 PB available. Default 200GB/person.
- No backups (but a reliable system with RAID)
- Has dedicated tools for fast access, e.g. optimized find function 'lfs find'.

https://wiki.aalto.fi/display/Triton/Data+Storage
https://wiki.aalto.fi/display/Triton/Data+storage+on+the+Lustre+file+system

Triton local storage

Storage local to compute node

- `/tmp` -directory
- Dedicated: Best for calculation time storage
- Copy relevant data after computation, will be deleted after job completes
- `$TMPDIR` variable defines a temporary directory for the job

Ramdisk for fast IO operations

- Special location, similar to /tmp
- `$XDG_RUNTIME_DIR` -directory (20GB per user)
- Extremely fast (it’s just RAM) but small and temporary
- Use case: job spends most of its time doing file operations on millions of small files.

Triton special storage

Solid state drive servers

- Currently special server – not user accessible
- Good for random access
- Currently planning future use – if you have use case, let us know

- **Department filesystems (`/m/$dept/{project,archive}`)**
  - Not actually part of Triton – provided by Departments/Aalto
  - Not highly parallel
  - Not mounted on all the nodes
  - Mounted for convenient data transfer only
    - You must move computation data to scratch

File systems: Summary

Location	Type	Usage	Size/quota
/home	NFS	Home dir	1 Gb
/tmp	local	Local scratch	~800Gb (varies)
$WRKDIR	Lustre	Personal work	200GB default
/scratch/$dept/$project/	Lustre	Shared work	As needed
$XDG_RUNTIME_DIR	Ramdisk	Local scratch	20GB

Quotas

- Quotas limit how much space you can use (*and* how many files)
- Check with “quota” command
- Home: 1GB
- Lustre
  - Work: 200GB and increased as needed, project quotas as needed
  - Quotas and advisory, and are always increased as necessary as long as you manage data well
  - “‘Disk quota exceeded’ error but I have plenty of space”: a common problem. Caused by limitation of Lustre, see the wiki page.
- https://wiki.aalto.fi/display/Triton/Triton+Quotas

How to optimize IO

File system performance metrics

- Stream I/O and random IOPS
  - Stream measures the speed of reading large sequential data from system
  - IOPS measure random small reads to the system – number of metadata/block data accesses
  - To measure your own application, profiling or internal timers needed
  - Rough estimate can be aquired from /proc/<pid>/io or by using strace

Performance metric examples

- Total numbers

Device	IOPS	Stream
Sata disk (7.2k)	50-100	50 MB/s
SSD disk	3000-10 000	500 MB/s
Ramdisk	40 000	5000 MB/s
Triton NFS	300	300 MB/s
Triton Lustre	100 000	30000 MB/s

- Per jobs, with 200 concurrent jobs using storage...

Device	IOPS	Stream
Sata disk (7.2k)	50-100	50 MB/s
SSD disk	N/a	N/a
Triton NFS	1.5	1.5 MB/s
Triton Lustre	500	150 MB/s

- DON'T run job jobs from HOME! (NFS)

How to optimize IO/data?

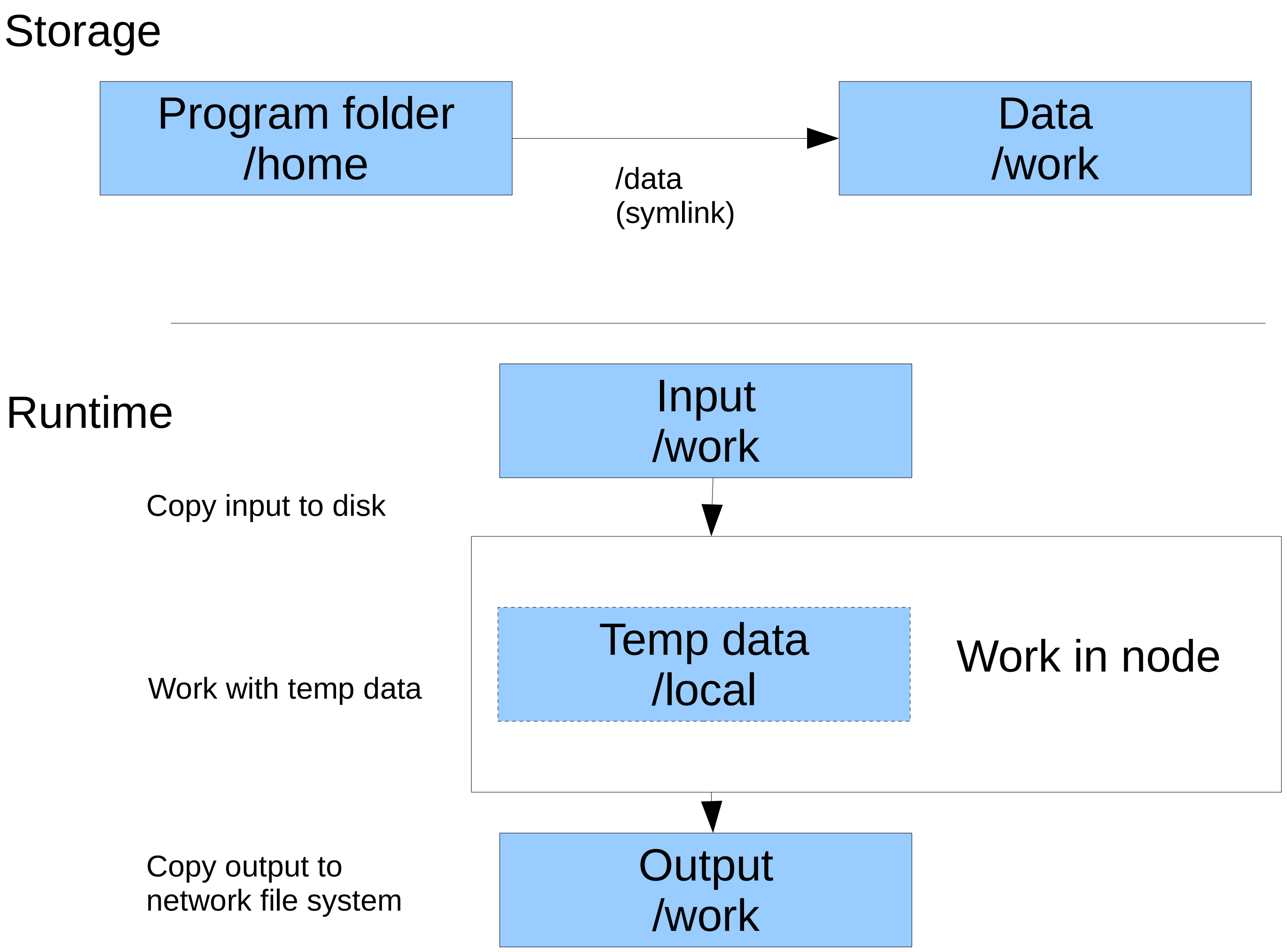
- Know how your program does its data handling, know which file system your program utilizes for its IO
- Measure your program with profilers e.g.
  ```
  strace -c -e trace=file <program>
  ```
- Minimize the number of unnecessary file calls e.g. log output timestep
- Load data in good sized chunks
- Do not do metadata calls unless they are necessary, access blockdata directly
- Save data in good formats with plenty of metadata

Advanced Lustre

- By default striping is turned off
  - “`lfs getstripe <dir>`” shows striping
  - “`lfs setstripe -c N <dir>`” stripe over N targets, -1 means all targets
  - “`lfs setstripe -d <dir>`” revert to default
- Use with care. Useful for HUGE files (>100GB) or parallel I/O from multiple compute nodes (MPI-I/O).
- Real numbers from single client (16 MB IO blocksize for 17 GB):

Striping	File size	Stream Mb/s
Off (1)	17 GB	214
2	17 GB	393
4	17 GB	557
max	17 GB	508
max	11 MB	55
max	200 KB	10

Workflow suggestion



Storage

Program folder
/home

/data
(symlink)

Data
/work

Runtime

Copy input to disk

Input
/work

Work with temp data

Temp data
/local

Work in node

Copy output to
network file system

Output
/work

Do's and Don'ts

Do's and don'ts: lots of small files

Lots of small files (+10k, <1MB)

- Well, bad starting point already in general. Though, sometimes no way to improve (e.g. legacy code)
  - `/ramdisk` or `/local`: Best place for these
  - Lustre: Not the best place. With many users local disk provides more IOPS and Stream in general
  - NFS (Home): Very Bad idea, do not run calculation from Home
- **The very best approach:** modify you code. Large file(s) instead of many small (e.g. HDF5). Or even no-files-at-all. Sometimes IO due to unnecessary checkpointing.

Do's and don'ts: inefficient `ls`

"**ls**" vs "**ls -la**"

- ls in a directory with 1000 files
  - Simple `ls` is only a few IOPS
- `ls -la` in a directory with 1000 files
  - Local fs: 1000+ IOPS (stat() each file!)
  - NFS: a bit more overhead
  - Lustre (striping off) 2000 IOPS (+rpcs)
  - Lustre (striping on) 31000 IOPS! (+rpcs)

    => Whole Lustre stuck for a while for everyone
- Use "`ls -la`" and variant (`ls --color`) ONLY when needed

Do's and don'ts: small files

500Gb of data

- Estimated read time in minutes

	1M Many small files	Single big
/local	170+	28
/scratch (stripe off)	170+	28
/scratch (stripe max)	BAD IDEA	8

- Use `/local` or Lustre (+ maybe striping) for big files
- Note that above Triton results assume exclusive access (reality: shared with all other users)!

File systems: Do's and Don'ts

Databases (sqlite)

- These can generate a lot of small random reads (=IOPS)
  - `/tmp` or ramdisk: Best place for these
  - Lustre: Not the best place. With many users local disk provides more IOPS and Stream in general
  - NFS (Home): very Bad idea

Best practices

- When unsure what is the best approach
  - Check above Do's and Don'ts
  - Google?
  - Ask your local Triton support person
  - Triton issue tracker and ask!
  - Ask your supervisor and colleagues
  - Trial-and-error (profile it)

Further topics

These are not covered here. Ask/Google if you want to learn more.

- Using Lustre striping (briefly mentioned)
- HDF5 for small files
- Benchmarking, what is the share of IO of a job
- MPI-IO
- Hadoop

Exercise: File systems

? minutes to proceed, use wiki/google to solve
All scripts are in /scratch/scip/lustre_2017

Simple file system operations

- Use mkdir and ln to create a project like the one in the workflow example.
- Use “`strace -c`” to compare “`ls`” and “`ls -l`”, and “`ls --color`”. Compare output with eg. grep/diff. Try listing individual files, and also the directory `/scratch/scip/lustre_2017/many-files`.
- Copy `create_iodata.sh` to your data folder and run it to create sample data. Compare “`strace -c`” of “`lfs find $dir`” and “`find $dir`” searches to the directory.
- Copy `iotest.sh` to your test project folder and submit it with `sbatch`. What does the output mean?
- Try to convert the code to use `$TMPDIR`. Once you're sure it works, change “`ls`” to “`ls -l`”. Compare the results.
- Convert the code to use tar/zip/gzip/bzip2. Can you deduce anything from /proc/<pid>/io output?

Questions or comments regarding Triton file systems?

References:

- https://wiki.aalto.fi/display/Triton/Data+Storage
- https://wiki.aalto.fi/display/Triton/Compute+node+local+drives
- https://wiki.aalto.fi/display/Triton/Data+storage+on+the+Lustre+file+system
- https://wiki.aalto.fi/display/Triton/Triton+Quotas