UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

# Efficient Use of BioHPC Storage Systems

[web] portal.biohpc.swmed.edu
[email] biohpc-help@utsouthwestern.edu



*Updated for 2021-03-24*

# Outline

- Overview of BioHPC systems storage
  - Different filesystems (/home2, /work and /archive, /project)
  - Quotas
  - Backups
- Understanding input and output (I/O)
  - I/O patterns
  - Introduction to data and metadata
- Using storage effectively
  - Data migration between storage systems
  - Data compression
  - General good practices

## BioHPC storage systems – Quotas and Backups

| Mount point | Location/FS Type | Quota | Backup |
|---|---|---|---|
| /home2 | Network/Fast SSD | 50GB/User | Mirror backup twice per week |
| /project | Network/Lustre | 5TB/Group | No backup |
| /work | Network/GPFS | 5TB/User | Mirror backup once per week |
| /archive | Network/GPFS | 5TB/Group | No backup |

`/project` and `/archive` have no backup, PI can request backup

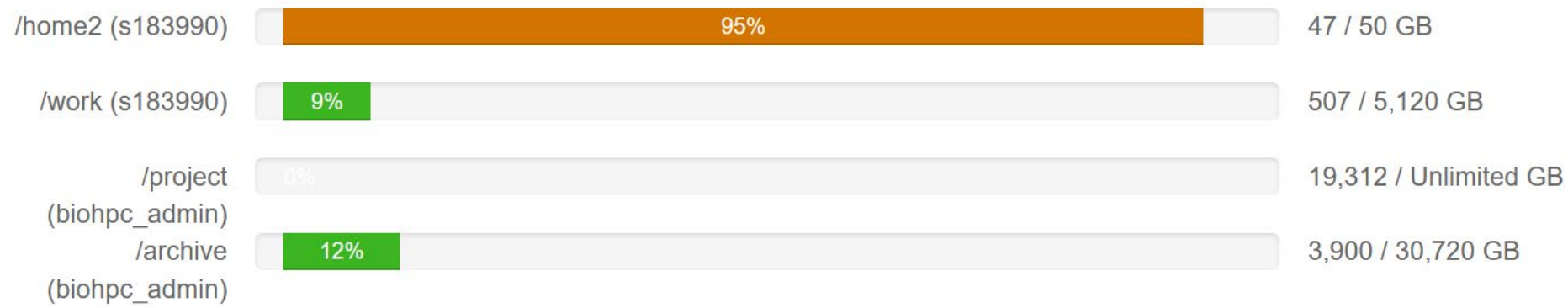
**/project and /archive can be increased on PI request with Dept. Chair approval.**
**/archive usage is multiplied by 2/3 (as to encourage use of archive).**
**Data on /home2 counts thrice and on work counts twice because of backup.**

## Storage Quotas

```
[s183990@Nucleus005 ~]$ biohpc_quota

Current BioHPC Storage Quotas:

  FILE    |          SPACE USAGE          |          NUMBER OF FILES
 SYSTEM   |     USED      SOFT      HARD  |       USED        SOFT        HARD
----------|-------------------------------|-----------------------------------
User quotas for s183990
----------|-------------------------------|-----------------------------------
home2     |   49109M    51200M    71680M  |        582k           0           0
work      |   507.5G        5T        7T  |      326251           0           0
----------|-------------------------------|-----------------------------------
Group quotas for biohpc_admin
----------|-------------------------------|-----------------------------------
project   |   18.86T        0k        0k  |    12710087           0           0
archive   |   3.809T       30T       40T  |     4176950           0           0
```

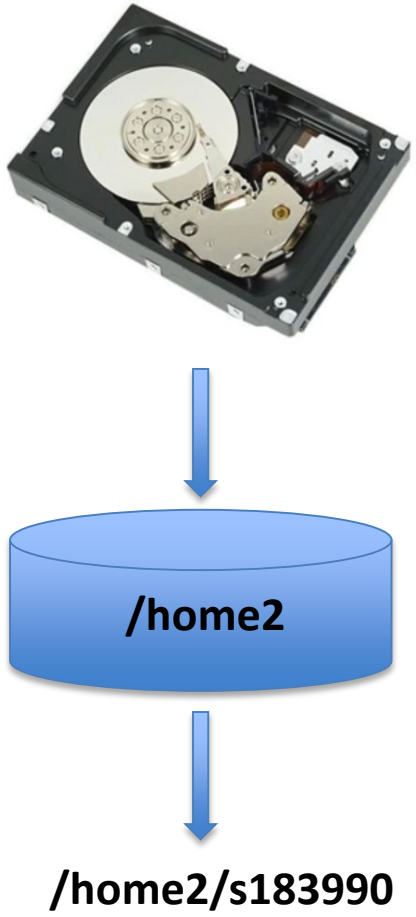


## A rough guide to choosing a storage system

- /home2
  - Small files that are very important, low I/O intensity
  - Not for files that  need to be shared with other users
- /work
  - Large amounts of files, high I/O intensity
  - Not for files that  need to be shared with other users
- /project
  - Large amounts of files, high I/O intensity
  - Good for files that need to be shared
- /archive
  - Large amounts of files, high I/O intensity
  - Good for files that need to be shared

Some popular filesystems:

- Windows: NTFS, FAT
- Mac/iOS: HFS+, APFS
- Linux: ext4, XFS
- **Parallel computing/HPC: GPFS, Lustre**



## How a filesystem organizes files and directories

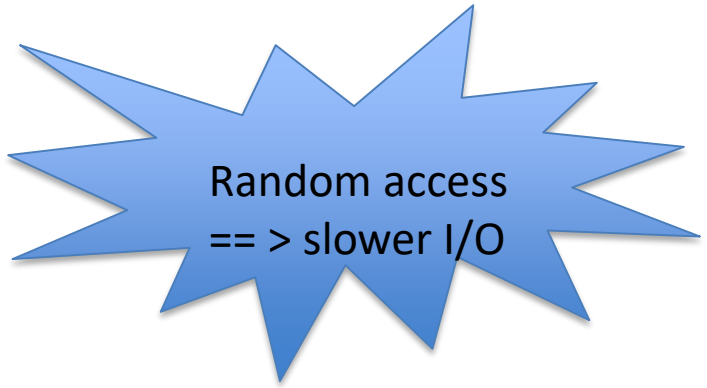
- A filesystem organizes files and directories into blocks of data
- The minimum size a non-zero file takes up on the filesystem:
  One block

Name: hello_world.txt
Owner: s183990
Group: biohpc_admin
Permissions: -rw-r--r--

Hello w | orld!

Types of access to data blocks:
- Sequential access
- Random access

## Random vs. sequential access




Simply, it is faster to write/read the same data with a single sequential I/O rather than multiple I/Os.

## I/O considerations while developing/running a workflow

- Size of the I/O affects the performance of your application
- Which do you think will be faster?


  - An application that reads 1000 MB sequentially by issuing 500 read requests of 2 MB each
  - An application that reads 1000 MB sequentially by issuing 50000 read requests of 20 KB each

## Data vs. metadata

- Data
  - A text file
  - A BAM file contains sequence alignments
  - A molecular dynamics trajectory contains atomic coordinates/velocities
  - An CT/MRI output contains an image of someone's organ.
  - An EM image contains a picture of a cell in the body
- Metadata: "Data about data"
  - When the file was created/accessed
  - The access permissions of the file
  - Where the data is located physically on the underlying disk

## More about metadata

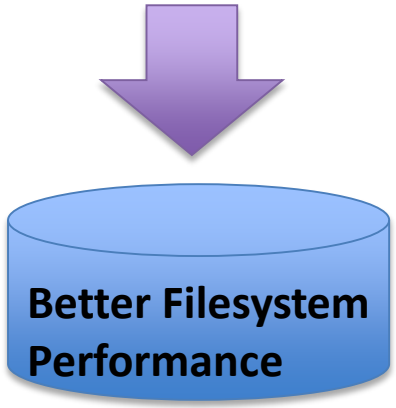
- Metadata is stored inside special disk blocks called i-node
- Limited, pre-defined metadata space
- Has pointers to the blocks on a disk that hold the data
- i-nodes are stored on SSDs

```
[s183990@Nucleus005 ~]$ stat aa.txt
  File: 'aa.txt'
  Size: 631             Blocks: 8          IO Block: 1048576 regular file
Device: 28h/40d Inode: 45125932468  Links: 1
Access: (0740/-rwxr-----)  Uid: (183990/ s183990)   Gid: ( 1001/biohpc_admin)
Access: 2021-03-02 16:14:30.445185292 -0600
Modify: 2019-06-10 15:08:55.607813979 -0500
Change: 2019-06-10 15:08:55.607465424 -0500
```

## Considerations about metadata

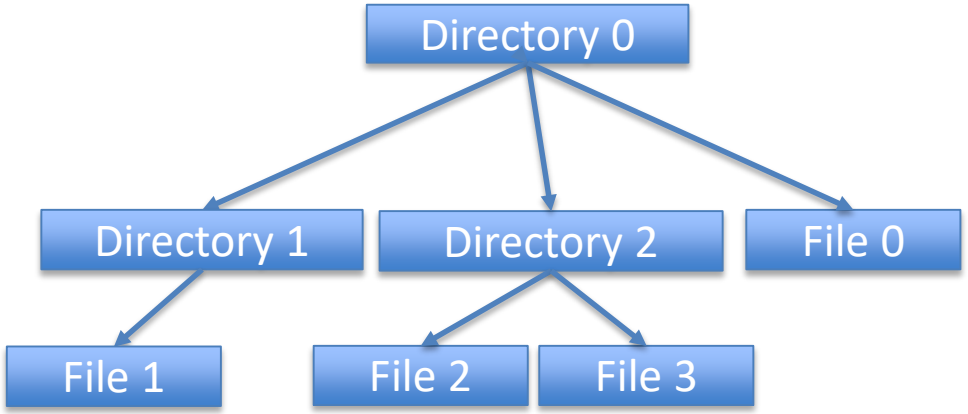
- Metadata is a shared resource: All users having access to a filesystem access the same pool of metadata
- Metadata heavy workloads ➔ slow down filesystems for everyone!

Things to avoid:
- Creating lots of files
- Large number of directory listings



## How directories are stored in the filesystem

- Directories are special files that hold pointers (links) to other files
- The more files there are in a directory, the larger amount of space the directory blocks will take up on disk
- Operations such as listing directories and moving files require operations on the directory blocks
  - The file system has to iterate through files in the directory individually
  - Bigger directories => longer operation times
  - Our recommendation: Having not too many files in a directory (not more than a couple of thousands ideally!)



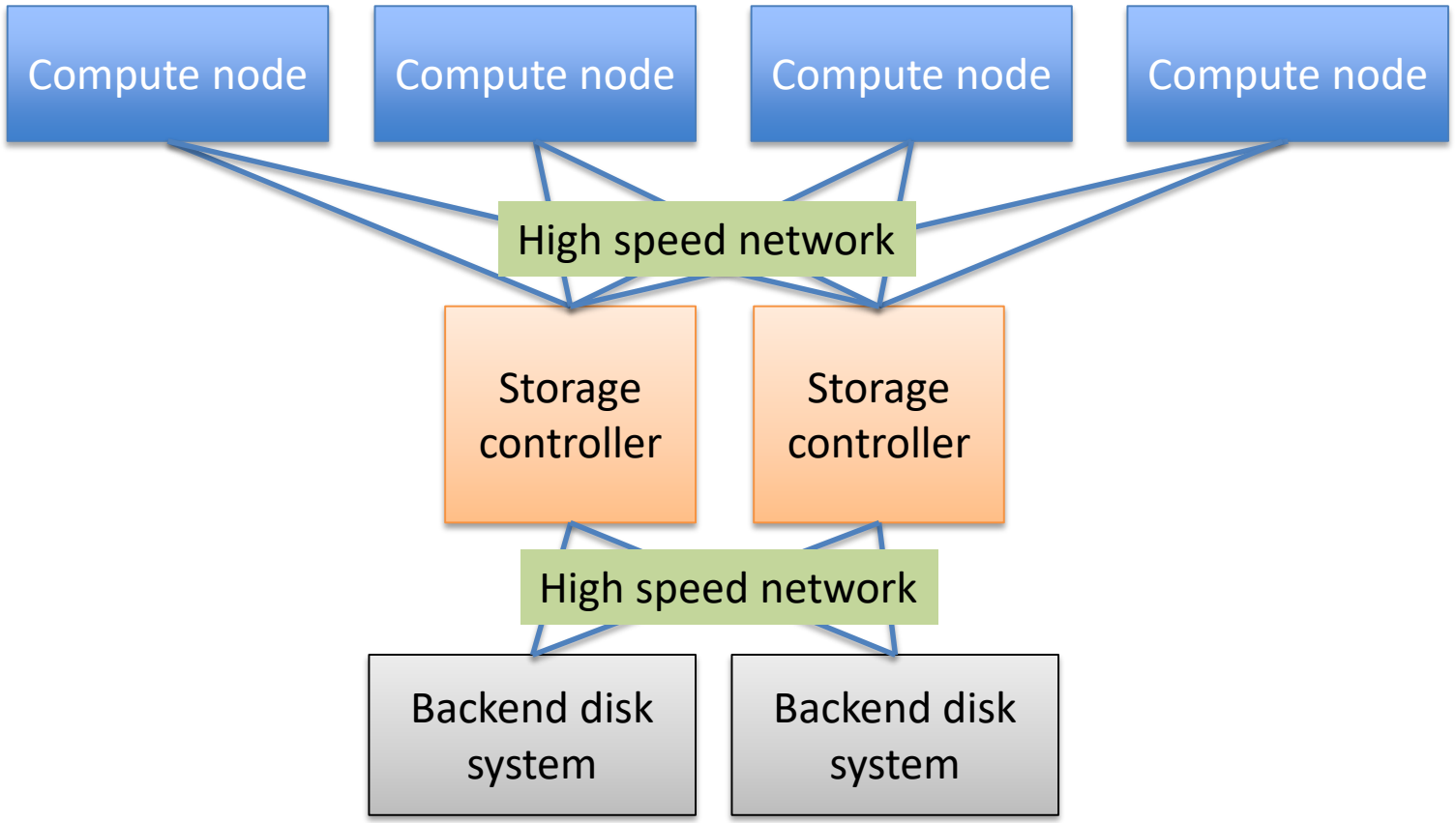
## General overview of parallel storage architecture

Main bottlenecks:

- Network
- CPU, memory, I/O on the controller

How to avoid bottlenecks?

- Avoid many parallel I/O operations
- Perform I/O on large blocks
- Avoid excessive metadata operations



Good practice:

- Reading a data file in once at program initiation, and then keeping the data cached in memory.

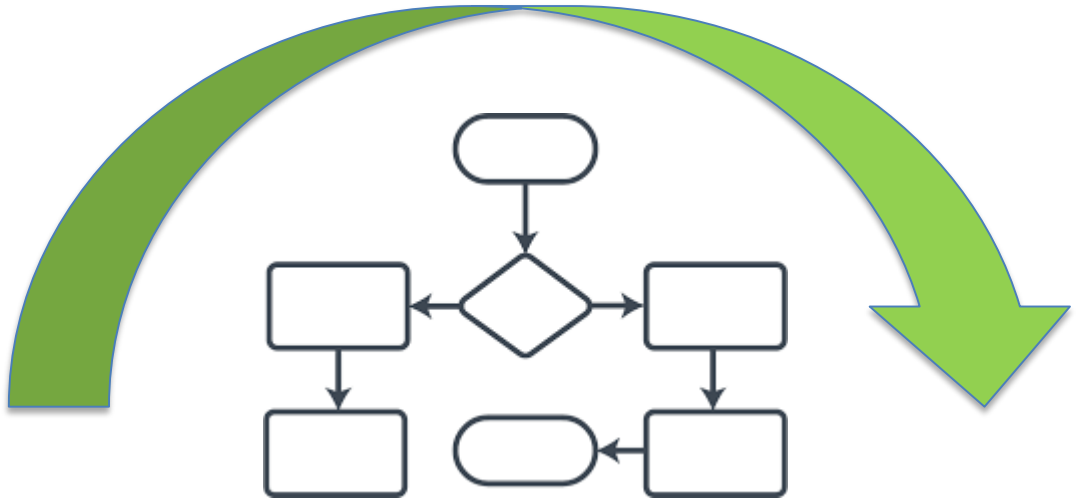
VS.

Bad practice:

- Having 1,000,000 data files in a single directory.

## Refactoring a workload (for developers)

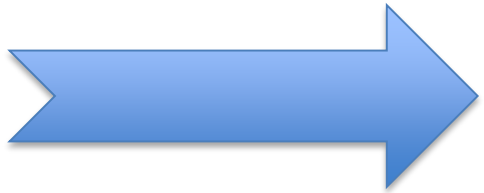
- Identify parts of the code where lots of parallel processes are doing I/O
    - Could one process do I/O and communicate with other processes ?
    - Can some or all of that I/O use /tmp directory on a compute node rather than the networked storage?
- Identify bottlenecks in the workflow
    - E.g. the whole workload has to wait until one file is updated



## Refactoring a workload (for developers)

**How to fix the following pipeline?**

- Lots of read and writes back to network storage.



- Only writes to networked storage when needed.
- Only write out final result files.

## Signs of inefficient use of the storage system

- When submitting a job, slow access to working directories involved with the job
- For parallel jobs, job completion speed was acceptable for small numbers of jobs, but gets dramatically slower as the size or number of jobs increases.
- BioHPC staff will monitor the storage systems and notify you if your job(s) having a negative impact

## Using space efficiently

- Move data from /project to /archive when processing is done
- Delete any raw data or intermediate files that you're sure you won't need again (or that are backed up elsewhere)
- Compress (bzip2, etc.) files when not in active use
- Shared directories

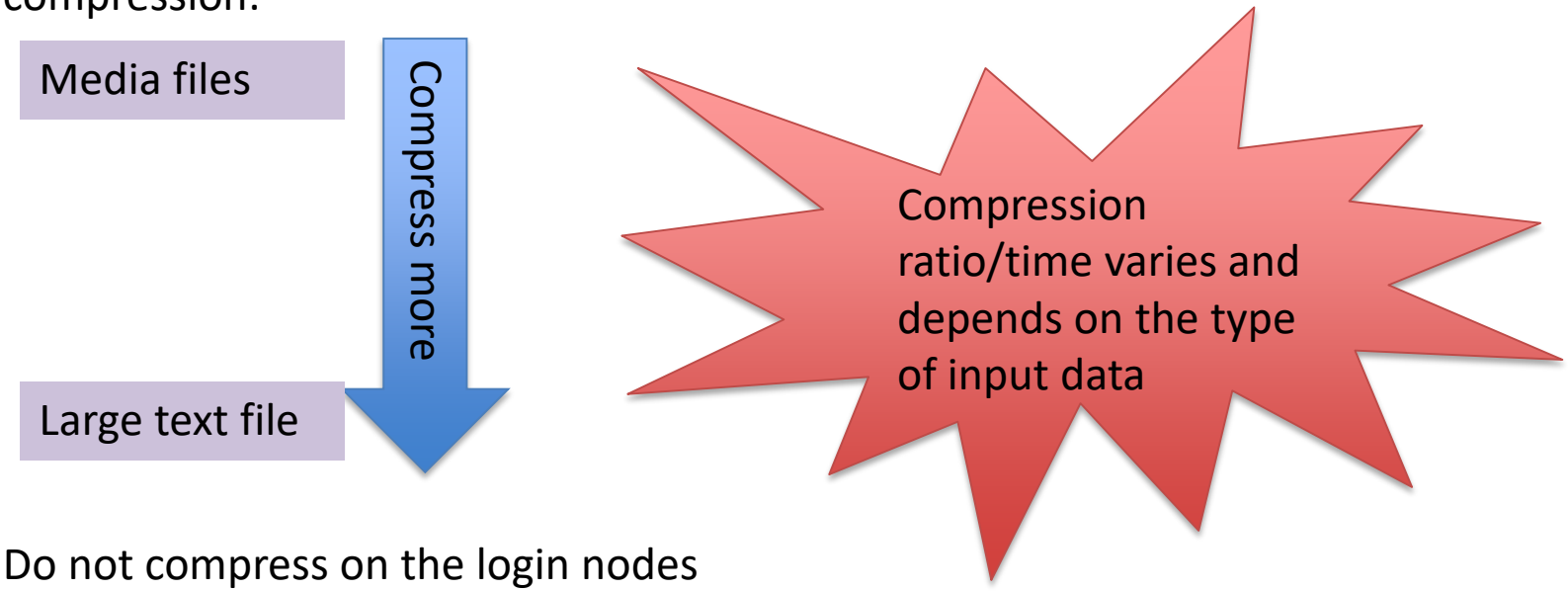
## Compression techniques

- **bzip2** is a popular **lossless** compression technique
- As a rule of thumb, more potentially efficient is a compression algorithm, more CPU it requires.

- Create `.tar` archive file: `tar cvf dir2.tar /dir1/dir2`
  - **c** – Creates a new .tar archive file
  - **v** – Verbosely show the .tar file progress
  - **f** – File name type of the archive file
- Create a `.tar.bz2` archive file: `tar cvfj dir2.tar.bz2 /dir1/dir2`
  - **j** – Creates a highly compressed file
- Using bzip2:
  - `bzip2 file_to_compress`

## Compression considerations

- Not all files are compressible or have the same level of compression:




- Do not compress on the login nodes
- Do not compress files which have already been compressed
- Try to use a parallel compressor `(pigz)` since it will lower compression times (more on that later on).

## Example of parallel compression using pigz

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --partition=super
#SBATCH --cpus-per-task=20
#SBATCH -J compress
#SBATCH -o comp.%J.out
#SBATCH -e comp.%J.err

# Compressing with pigz:
module load pigz/2.4
pigz -9 /project/biohpcadmin/s183990/files_to_compress/*
```

**6** – Default compression.
**1** – Fastest but offers the least compression.
**9** – Slowest but the best compression.

## Moving data between directories

- **`rsync`** is preferred in the case of
  - incremental transfer,
  - remote destination,
  - and unreliable communication

More about **`rsync`**

- Designed for efficient transfer of data across networked computes/storage systems
- Written in C
- Single threaded
- Used for minimizing network usage
- Uses SSH for remote data transfer:
  - **`rsync local-file user@remote-host:remote-file`**

## Examples using rsync: single-threaded

- Before running **`rsync`**:
  - Make sure there is enough space in the destination filesystem
  - Allocate a 32GB node on the cluster through Slurm or the web visualization
  - Choose the right flags for the **`rsync`**

```
rsync -aAvh /project/biohpcadmin/s183990/ /archive/biohpcadmin/s183990/project_bak/
```

-a: archive mode
-A: preserve ACLS
-v: increase verbosity
-h: output numbers in a human-readable format

50 – 100 Mbytes/sec

## Examples using rsync: multi-threaded

Use multi-threading for better performance using **xargs**

```
ls /project/biohpcadmin | xargs -n1 -P4 -I% rsync -avh /project/biohpcadmin/s183990/% /archive/biohpcadmin/s183990/project_bak/%
```

Use **nohup** to run the job in the background mode:
$ **nohup** [long rsync command] **> nohup.out** followed by Ctrl + z
$ bg (make the process to run in the background)
$ ps –ef | grep rsync (checks the process status)

Parameters affecting the **rsync** performance
- Number of files inside the source directory
- Load on the filesystems

## Putting it all together once more!

Efficient use of storage space

- Sore data in /archive as much as possible – it's the cheapest!
- Consider removing data if not used in a long time (6 months?)
- Perform data compression in case do not plan to remove data

Application efficiency

- Do some benchmarks regarding the I/O of your application
  - E.g. try a range of numbers for reading/writing files/blocks

Filesystem efficiency

- Do not store a large number of files inside any of the filesystems
- Do not perform metadata operations such as `ls` on directories with large amounts of files

Contact BioHPC-Help regarding any storage related question!