FROM RESEARCH TO INDUSTRY

www.cea.fr

Isolating failure domains using OST pools

LAD'17

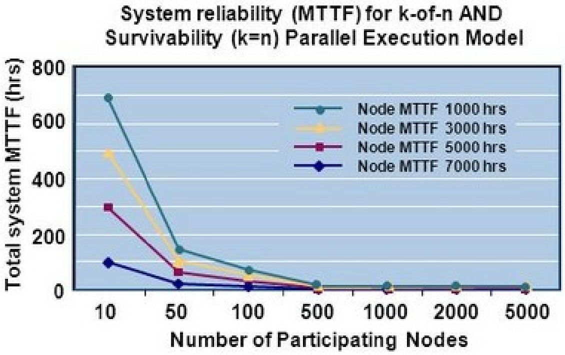
Thomas Leibovici <thomas.leibovici@cea.fr>

October 4-5 2017

Prepare for failures!

More components = more failures

- Lustre's strength is its scalability
  - Allow aggregating throughput of many disks, servers, network links...
- The more components, the higher the failure probability
  - MTBF of components is not infinite
  - High concurrency triggers software bugs more likely



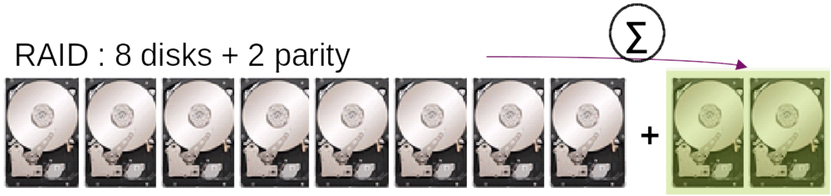
Modeling of reliability in HPC (Stephen L. Scott, ORNL)

=> Failure is the norm in a large systems

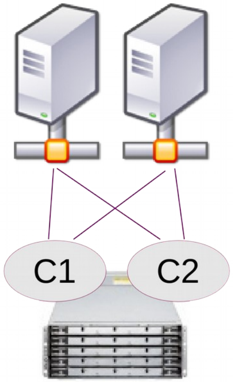
Commons ways to prevent failures

Common redundancy solutions

- **RAID** protects against:
  - Block corruption
  - Disk failure



- **Dual controller/dual attachment** protects against:
  - Disk array controller failure
  - Damaged link



- **HA** protects against:
  - Server failure
  - Network adapter failure
  - Software failure (e.g. LBUG)

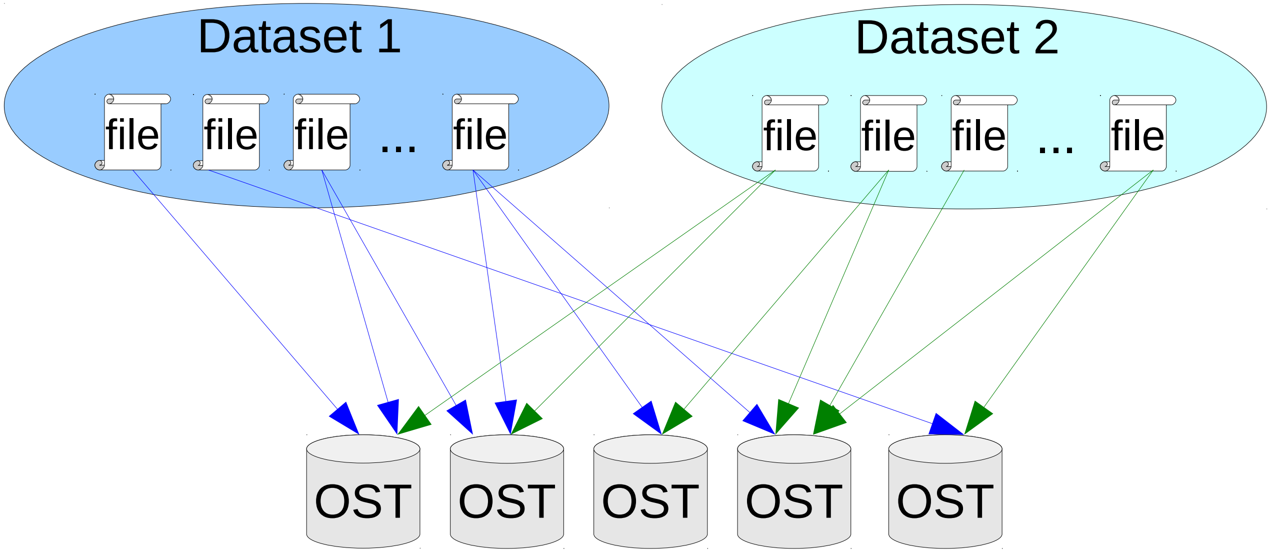
Big problems when larger failures occur

- Loss of more disks than parity count
- Whole disk array failure (e.g. double controller crash)
- HA failure

Why striping make it worse

Default = stripes anywhere

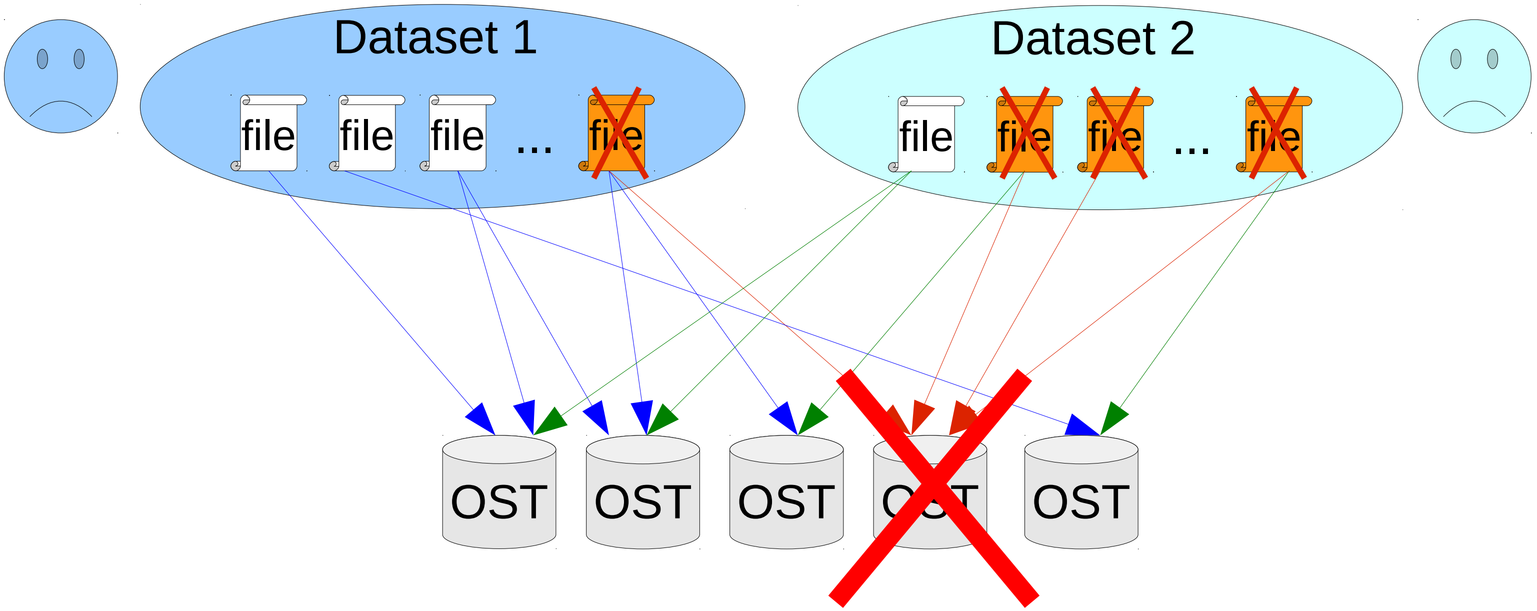
- Lustre default striping only relies on OST usage and load balancing
  - User's data is everywhere
  - If any OST becomes inaccessible, most datasets are impacted
  - Partial datasets are often unusable



Why striping make it worse (2)

Default = stripe anywhere

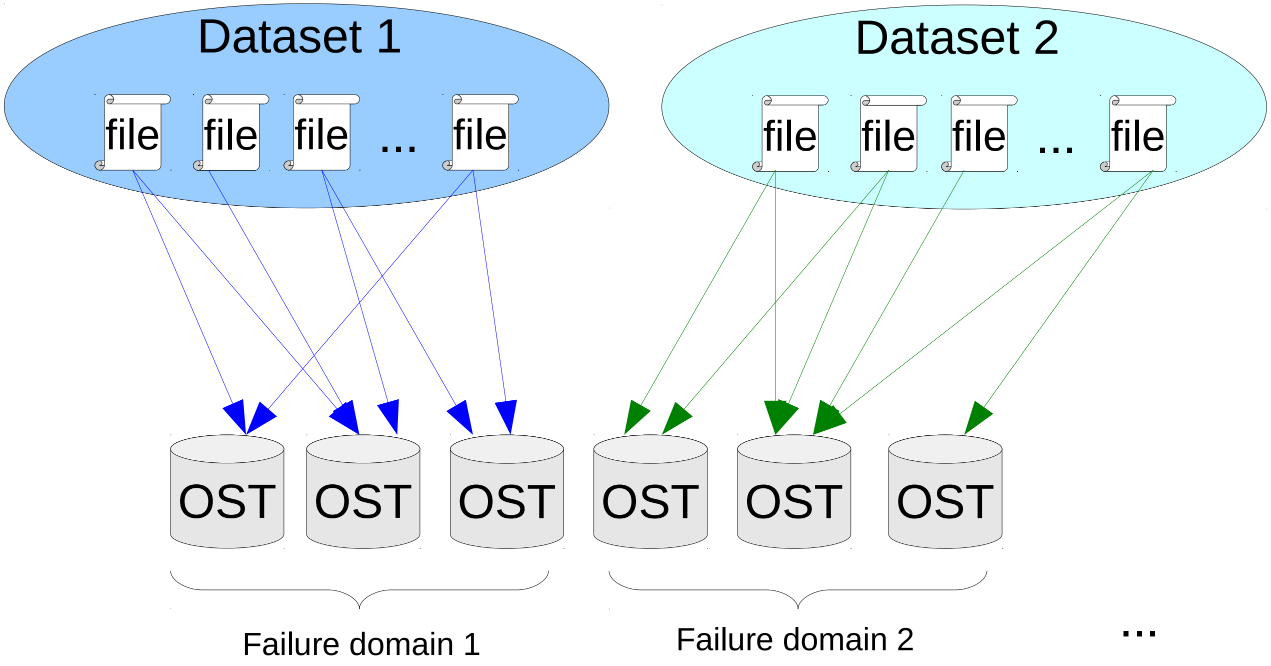
- Lustre default striping only relies on OST usage and load balancing
  - User's data is everywhere
  - If any OST becomes inaccessible, most datasets are impacted
  - Partial datasets are often unusable



Grouping stripes into “failure domains”

Why grouping stripes?

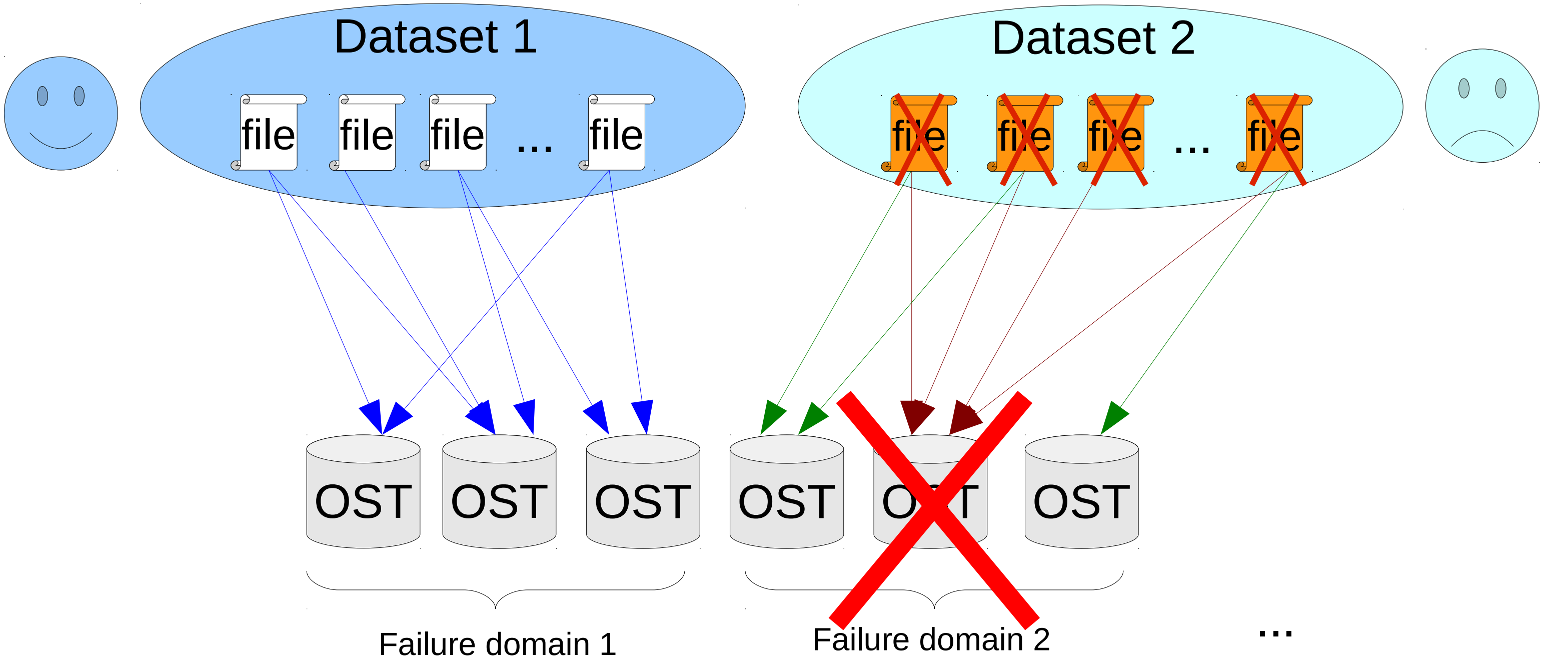
- Grouping datasets in failure domains reduce the number of impacted datasets
  - In case of OST failure, most datasets remain available
  - E.g. 1 failure domain = 1 HA Cell



Grouping stripes into "failure domains" (2)

Why grouping stripes?

- Grouping datasets in failure domains reduce the number of impacted datasets
  - In case of OST failure, most datasets remain available
  - E.g. 1 failure domain = 1 HA Cell



OST pools feature

How to group stripes?

- OST pools allow creating logical groups of OSTs

```
lctl pool_new fs1 da3
lctl pool_add fs1.da3 fs1-OST[0-4f]
```

- Pool can be assigned at file creation

```
lfs setstripe -p fs1.da3 /fs/home/foo/my_study/my_file
```

- Pool can be assigned to directories

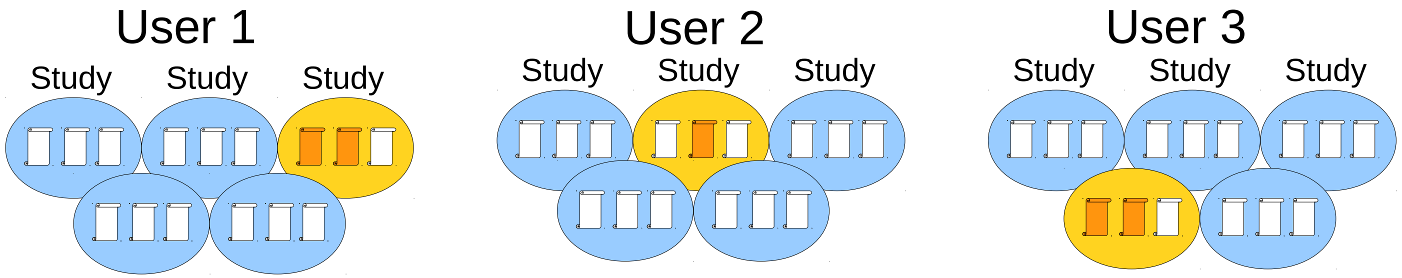
```
lfs setstripe -p fs1.da3 /fs/home/foo/my_study
```

  - Files inherit the pool of their parent directory
  - Sub-directories also inherit the pool of their parent directory
  - All “my_study” is located in the specified pool

Organizing data

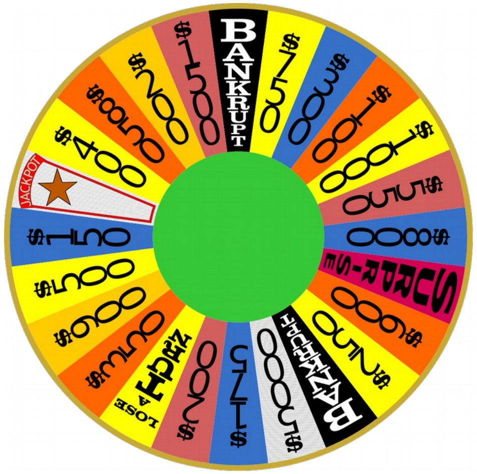
Defining the right datasets

- Per file: datasets of multiple files are unusable in case of OST failure
- Per user: some users loose access to all their data in case of OST failure
- Per group/community: even worse
- Per study/per compute job:
  - On case of OST failure, some studies are unavaible
  - Unavaible datasets are “fairly” spread between users
  - Most studies remain fully available
  - Every user/group/project still has full datasets to work on



Site integration

Assigning and turning pools



Solution 1

- Explicit set stripe when a study/compute job starts
  - Round-robin pool or random pool
    (avoid putting all user's eggs in one basket)

Solution 2

- Based on a common organization of user's tree

  e.g. `<user_dir>/<sub-project>/<job_dir>`
- Periodically (e.g. hourly), a system script changes pool assignment of all `<user_dir>/<sub-project>` directories (random or round-robin)
- Newly created job directories inherit from this pool

  => All data of a job is co-located on a pool
  => User's jobs are spread across pools

Other interests

Bonus

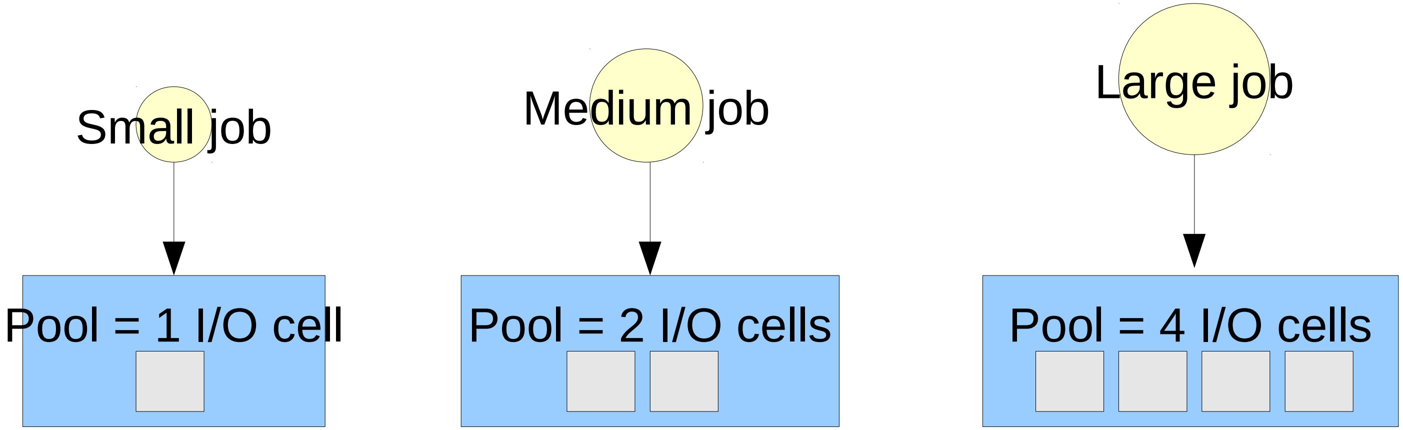
- In case of failure on some OSTs, production flow is easy to control:
  - Stop assigning impacted pool(s) to user's directories
  - Assign new job directories to sane pools

- Not only useful for big failures:
  - It can also be used to reduce I/O load, to speed up RAID rebuild

Drawbacks

Scaling the bandwidth per job

- A job cannot use the full filesystem bandwidth
  - It is limited by the bandwidth of pool resources
- OK for many small or medium compute jobs
  - All jobs aggregated can use the full filesystem bandwidth
- Doesn't fit for huge computations that need the whole filesystem bandwidth
  - Possibility to define larger pools for large jobs



Reorganizing an existing fileystem

Re-organizing existing data

Robinhood v3 custom policy to group files in pools

- If you wish to group existing files in pools
- Define a “no_pool” fileclass, that consists of files to be relocated:

```
fileclass no_pool {
    definition { type == file and ost_pool == "" }
}
```

- Define a custom policy, e.g.:

```
define_policy move2pool {
    status_manager = basic;
    scope { type == file }
    default_action = cmd("migrate2pool.sh '/fs/.lustre/fid/{fid}'");
}
```

  - Script “migrate2pool.sh” decides in which pool to locate the file and execute (possibly remotely) a command like:

    ```
    lfs migrate -p <pool> <file>
    ```

    => Access-proof (and raceless) since Lustre 2.8 (or with patch of LU-4840)

Re-organizing existing data (policy rules)

- Finally apply the policy to “no_pool”:

```
move2pool_rules {
     rule set_pool {
         target_fileclass = no_pool;
         condition { last_access > 1h }
     }
}
```

- Or, a more complete example:

```
move2pool_rules {
     rule set_pool_small {
         target_fileclass = no_pool_small;
         action = cmd(“migrate_local.sh -p poolK -c 1 {path}”);
         condition { last_access > 1h }
     }
     rule set_pool_medium {
         target_fileclass = no_pool_medium;
         action = cmd(“migrate_remote.sh -p poolM -c 4 {path}”);
         condition { last_access > 1h }
     }
     ...
}
```

- Running the policy

```
robinhood --run=move2pool --target=all
```

Re-organizing existing data (monitoring progress)

Commands to monitor migration progress

- Remaining files to be relocated:

```
# rbh-report --class-info=no_pool

fileclass,     count,    volume,   spc_used,   min_size,   max_size,   avg_size
  no_pool,     49750, 577.59 TB, 577.12 TB,  80.59 MB,  906.09 GB,   11.58 GB
```

- Status of migration actions:

```
# rbh-report --status-info=move2pool

move2pool.status,     type,      count,     volume,   spc_used,    avg_size
               ,   symlink,        125,    8.01 KB,  420.00 KB,          66
               ,       dir,      71204,  461.71 MB,  463.00 MB,     6.64 KB
               ,      file,      15520,    2.18 TB,    2.18 TB,     4.29 GB
             ok,      file,     802931,    1.95 PB,    1.94 PB,     2.54 GB
         failed,      file,        812,  757.34 TB,  757.34 TB,     2.19 GB
```

Conclusion & perspectives

- Even with RAID and HA, tragic situations can occur
- The presented method makes it possible to keep your filesystem usable even in such cases
- Pool feature proved to be very convenient to achieve this (stable, met our expectations)
- Interest of using robinhood to move data between OST pools

- Perspectives:
  - Use pools to manage multiple storage classes in a single namespace: SSD pool, HDD pool...
  - Use similar robinhood policies to move data automatically between pools (e.g. hot data to flash, cold data to HDD)
  - Even more perspectives with PFL, FLR...

Thank you for your attention !

Questions ?

Commissariat à l’énergie atomique et aux énergies alternatives
CEA / DAM Ile-de-France | Bruyères-le-Châtel - 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00

DAM Île-de-France

Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019