DataDirect™
NETWORKS
INFORMATION IN MOTION™

File and filesystem fragmentation in Lustre

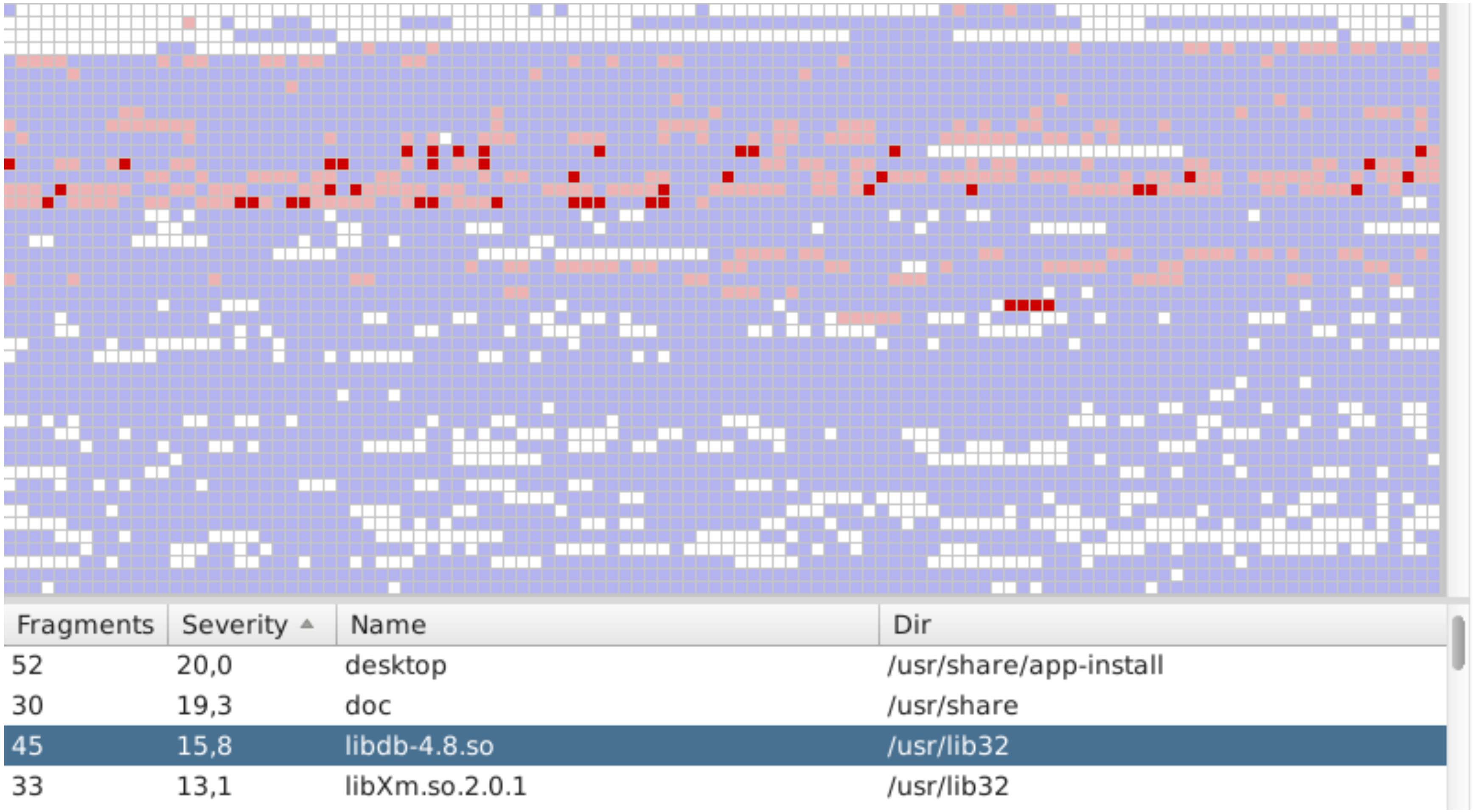
Ashley Pittman

apittman@ddn.com

What is fragmentation

- File fragmentation
  - Contents of individual file is dispersed over different locations on the device
- Filesystem fragmentation
  - Available space is dispersed over different locations on the device.

Fragmentation



Fragments	Severity ▲	Name	Dir
52	20,0	desktop	/usr/share/app-install
30	19,3	doc	/usr/share
45	15,8	libdb-4.8.so	/usr/lib32
33	13,1	libXm.so.2.0.1	/usr/lib32

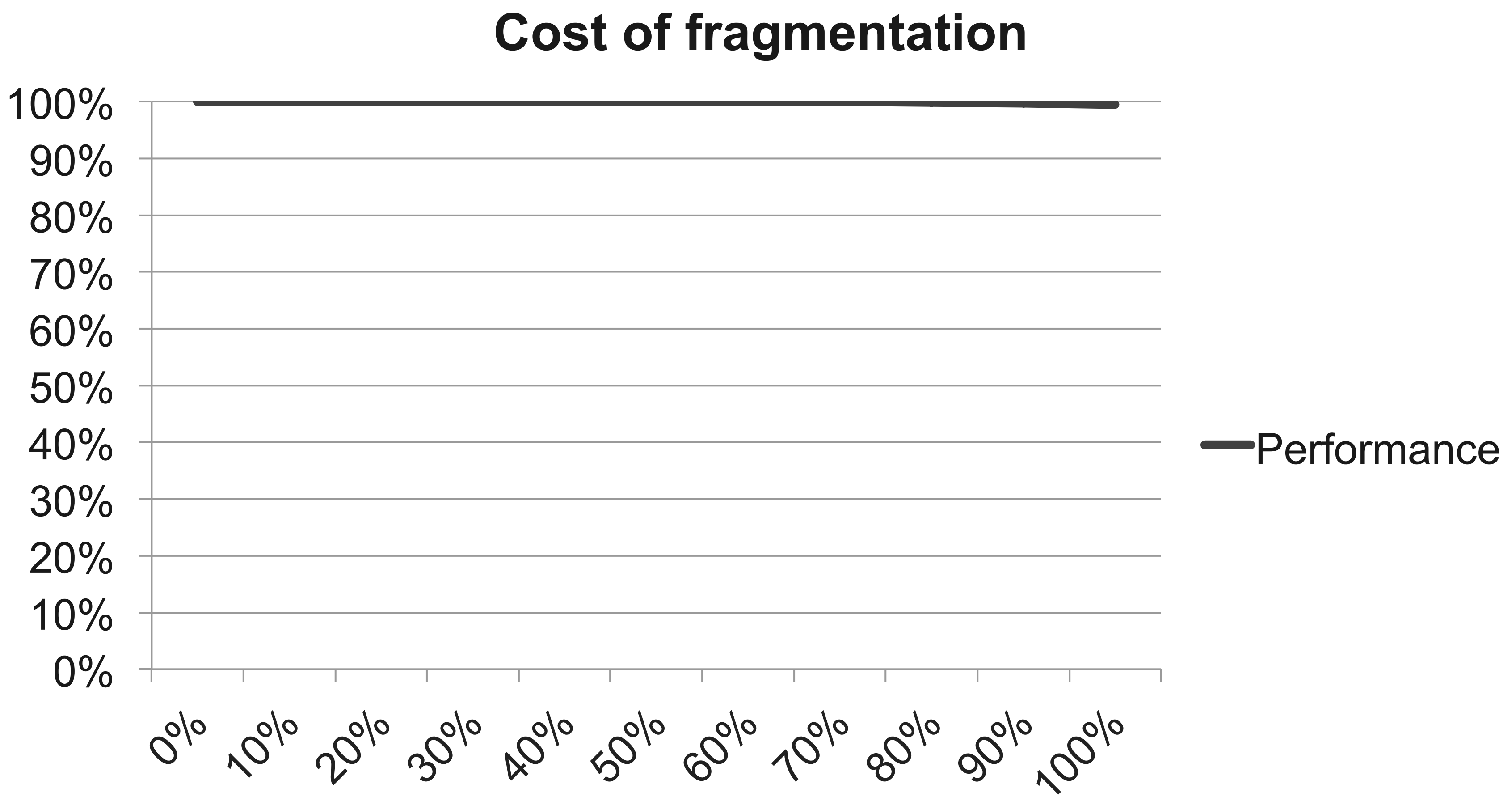
Why is this bad?

- Spinning media is good for streaming I/O
  - But poor for seeks.
- With file fragmentation seek performance becomes the factor in dominant I/O performance.

Assumptions

- Fragmentation cost is a function of utilisation level.
  - Appears to be the case
  - Will depend hugely on workload
- Cost of utilisation is not just fragmentation, but also the time cost of block allocator.

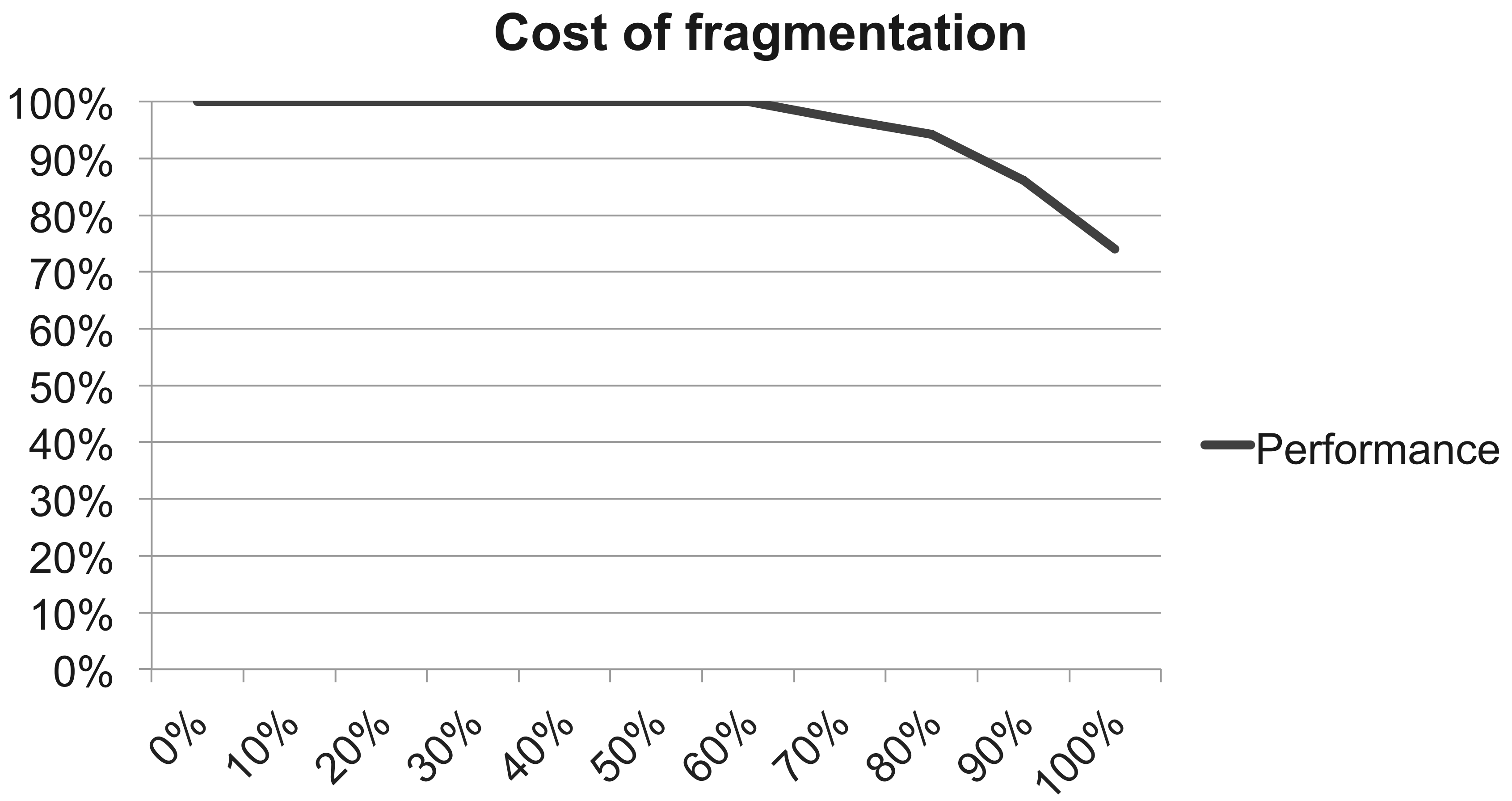
Single OST performance



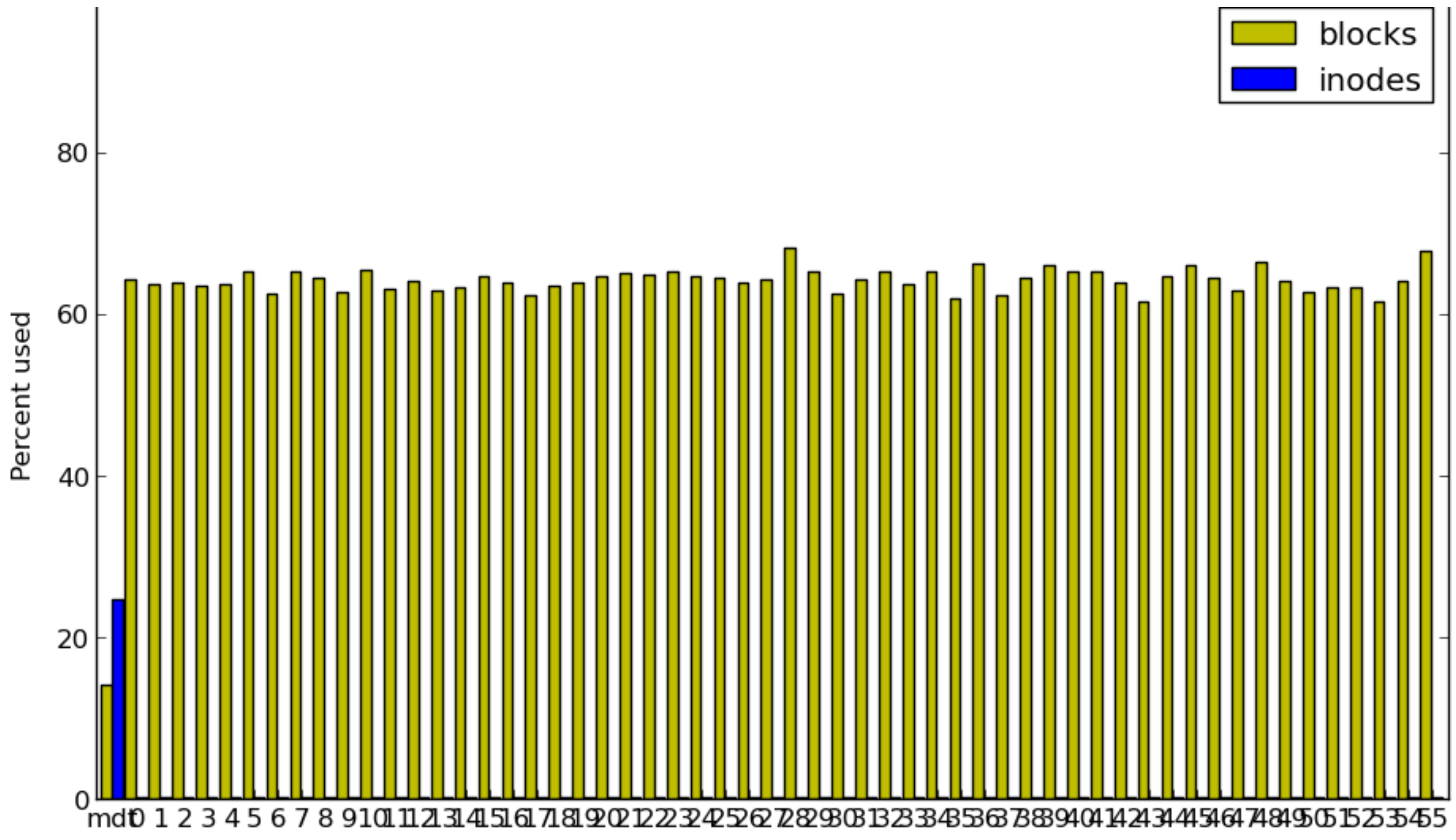
Why is this bad on Lustre?

- Parallel writes use many OSTs for performance.
- Performance is number of OSTs multiplied by speed of the slowest OST.
  - A single slow OST can have a dramatic effect on the overall bandwidth
- Likelihood of at least one OST being slow is probability of an individual OST being slow, raised to the power of the number of OSTs.
  - Increasing likelihood as OST counts rise.

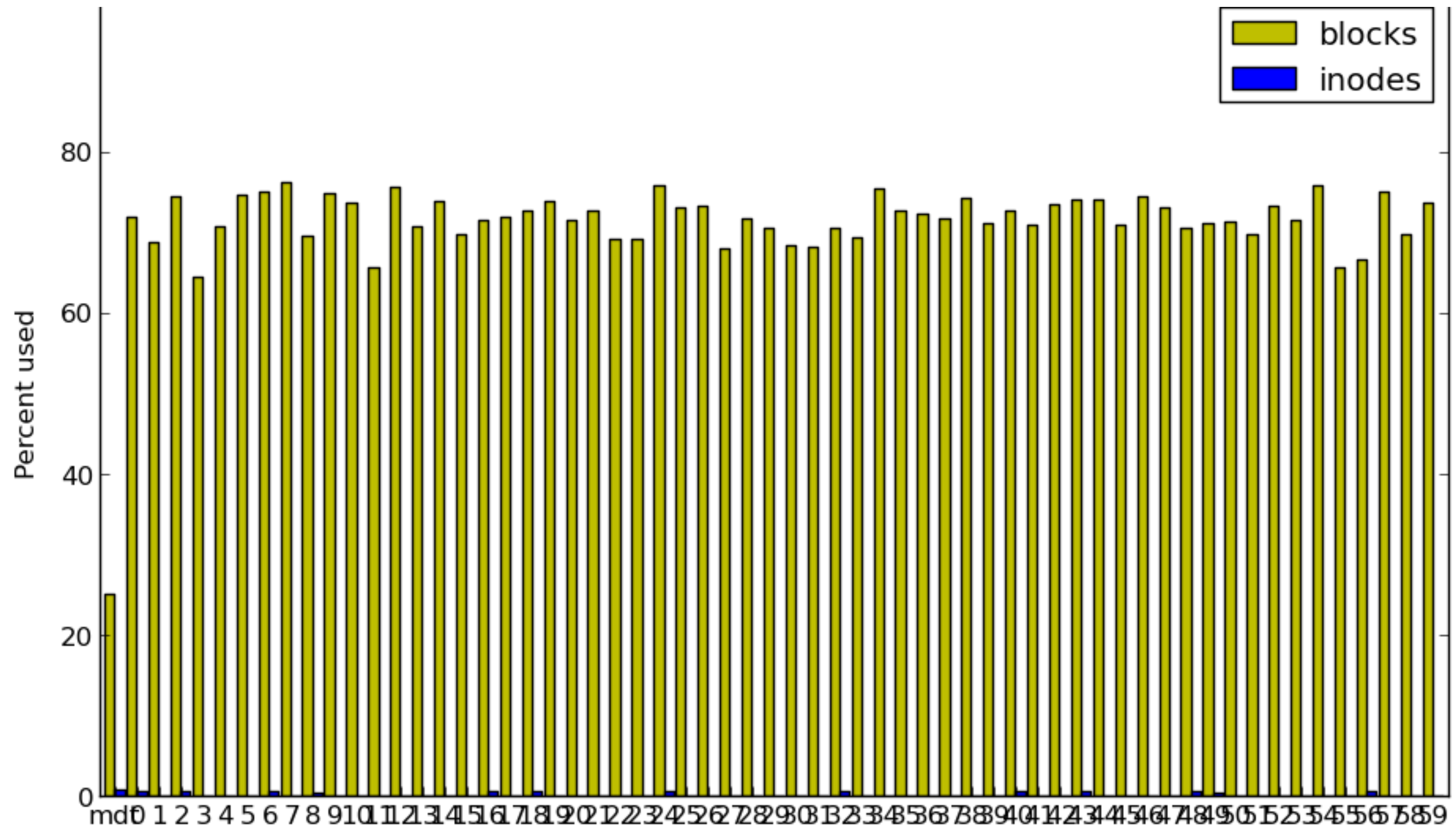
Parallel, 60 OST performance



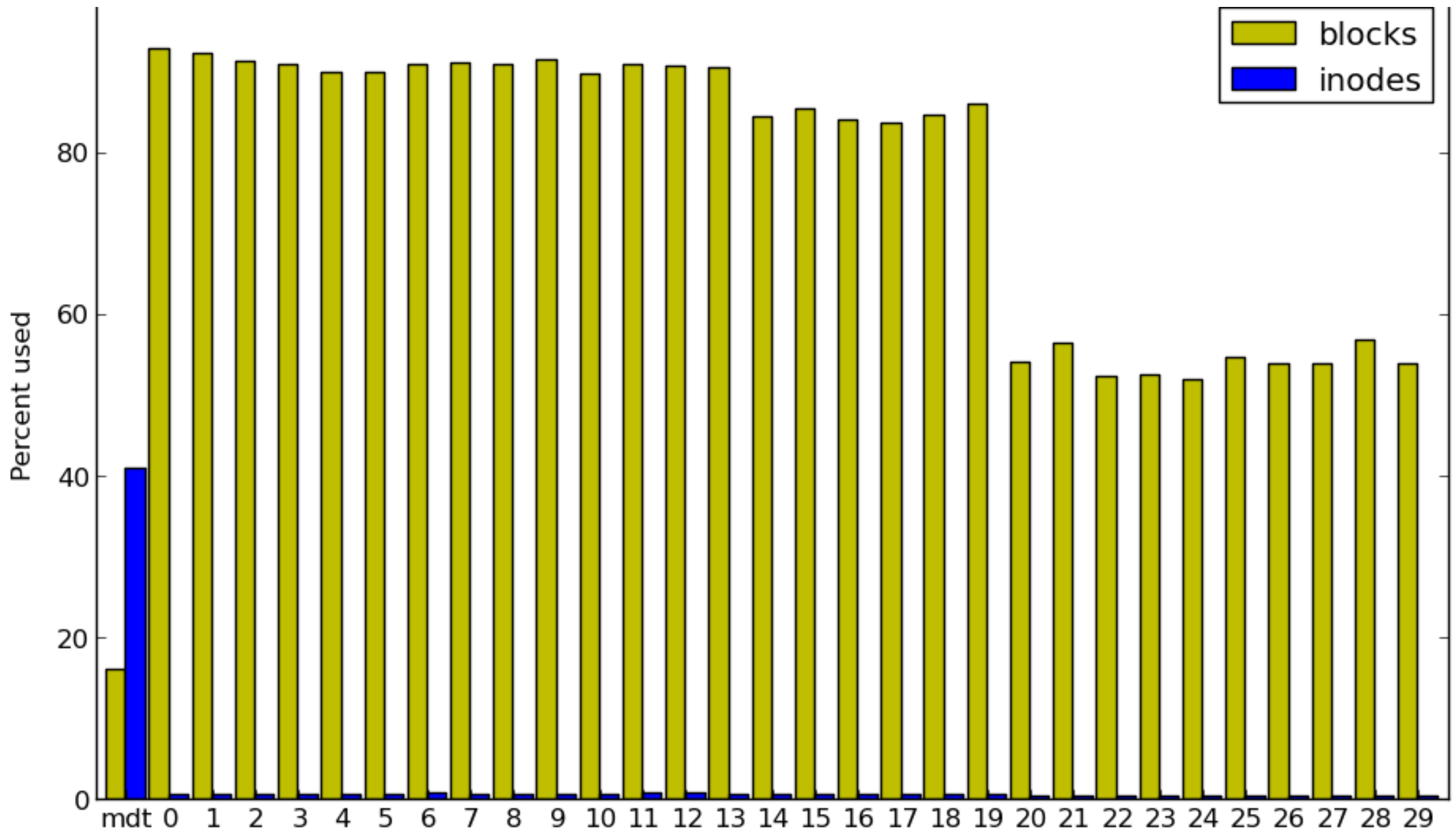
OST utilisation - good.



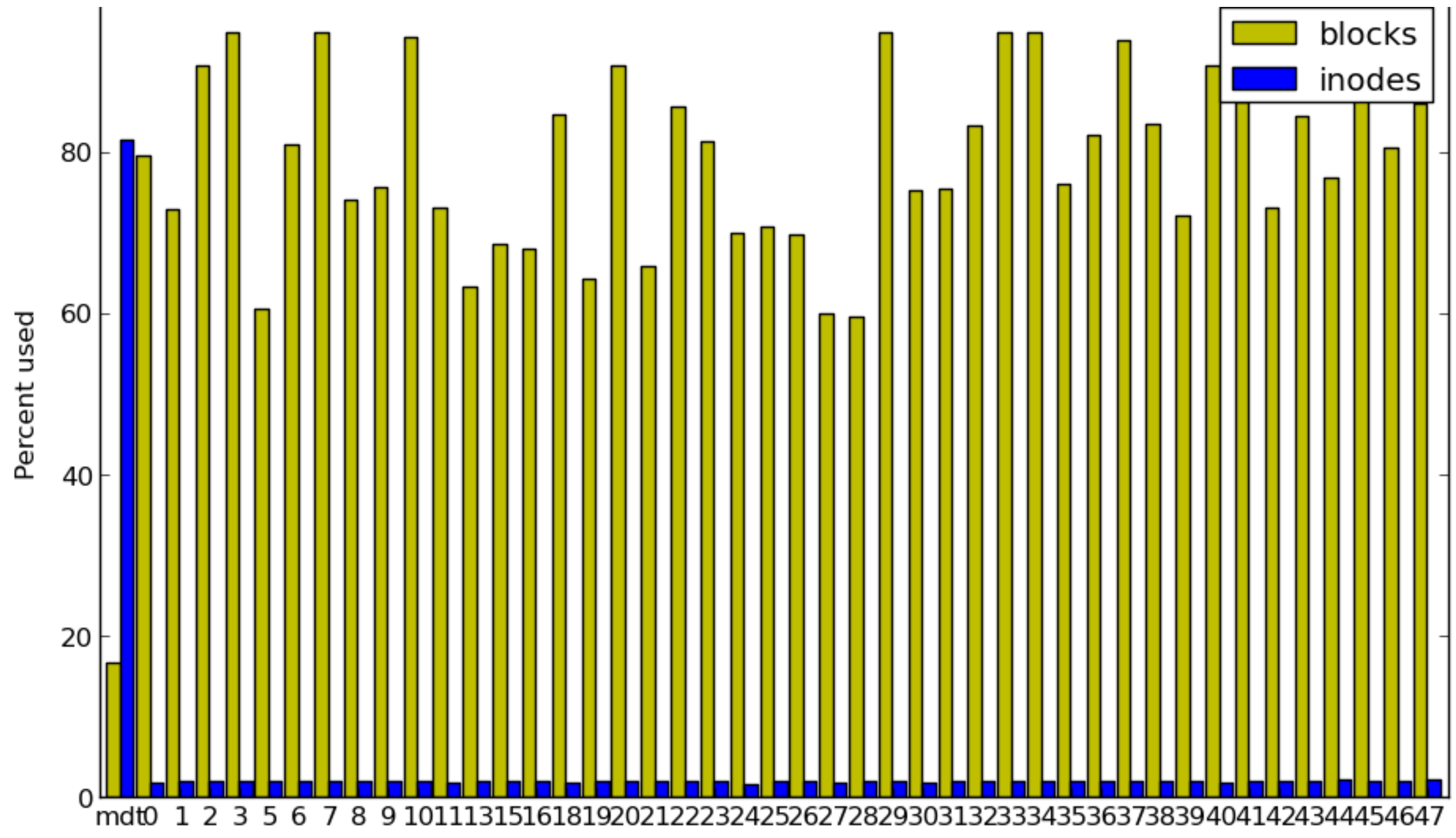
OST utilisation - good



OST utilisation - bad



OST utilisation - ugly



Quick solutions

- Rebalance files
  - Now possible with 2.4
  - Only works with adequate space available.
- Reduce usage levels

Avoidance tips

- Overspecify the filesystem.
  - Buy twice as much space, and use 100% SSDs.
- Don’t consume all the space
- Reduce individual OST fragmentation by limiting number of small files
- Keep OST space utilisation flat
  - Avoid, unstriped files.
- Larger block allocation sizes.
- Stripe to subset of OSTs?
  - Potentially avoiding overly-full OSTs so avoiding worst effects for more bandwidth.

Is read any better?

- Potentailly aoi_read() can avoid the issue.
  - Smaller reads can complete individually, allowing processing as the data arrives.
  - Adds significant complexity to application.

Hidden problems – existing files

- Historic OST fragmentation will lead to residual problems
- Hard to identify files
- Impossible to benchmark
  - Elusive but will affect wall-clock times.

Detecting problems

- filefrag –v <filename>
  - Shows block ranges used for files.
  - Can be used to discover if specific files are affected

Finding at-risk files.

- Large files
- Probably striped
  - If large and not striped possibly part of the problem.
- Specific creation date range
- List of candidate OST

Finding at-risk files.

- Large files
- Probably striped
  - If large and not striped possibly part of the problem.
- Specific creation date range
- List of candidate OST.

POSIX!

Conclusions

- Scaling costs are huge
- Best practice can avoid the issue in most cases
- Often un-diagnosed
  - Better monitoring and awareness
- Easy to diagnose
- Potential quick-fix for new files
- Slow-fix available for existing files
  - If you can find them.
- Block allocation is a major factor

What about ZFS?

- Different performance profile
  - Write policy
  - COW
  - Fewer OSTs

- Same basic theory applies

Questions?