Scratch Management and Scalable Flushing

Robert C. Bell | CSIRO IMT Scientific Computing
29 May 2018

Outline

- Scratch (temporary) shared filesystems
  - use and abuse, management
- Flushing – removal of “old’ files
  - Old CSIRO algorithm
  - New scalable algorithm

Temporary storage

- Scratch (temporary) shared filesystems
  - Often the biggest and highest performing FS on HPC systems
  - Provides temporary storage for the duration of jobs and sessions
  - Needs to hold some data for longer to support development, workflows, pre- and post-processing, analysis, etc.
- Shared – need policies for management
  - Many HPC sites have policies
  - Many sites do not have workable implementations

Goals

- Our approach is to manage scratch areas to try to maximize the use for the benefit of users, by allowing them to
  - gain access to large amounts of storage on demand (“campaign storage”)
  - store files in the scratch space for longer than individual jobs and sessions
  - reduce the copying to and from scratch
  - have large quotas
  - have automatic clean-up of old files
- Need to prevent scratch from filling

Space management - flushing

- Old CSIRO SC
  - scripts and program to implement policy
  - triggered when usage reaches a threshold (typically 95%)
  - file audit, then sort, and delete oldest until second threshold reached (typically 90%), or 7 days (rare)
  - uses mtime and atime – problem for FSes that don't do atime
  - also removes empty directories

New implementation of flushing at CSIRO

- Scratch areas shared between clusters and SGI UV 3000 (NFS)
- Old script slurped entire FS metadata into memory
- Scalability question – how quickly could the flushing respond under pressure?
- Known problems of metatdata performance on distributed FSes (NFS, Lustre)
- Servers (best place to run flushing script) did not have enough memory
- Problem with access time (upon which flushing is partly based) not being updated

- New policy – warning about inaccurate access times

New implementation of flushing at CSIRO

- Inspiration was gained from the article “It Probably Works”, McMullen, Tyler, CACM, vol 58, no 11, Nov 2015, pp 50-54.
- This article shows that we often do not need exact solutions, and can provide approximate solutions at far lower “cost”.
- Don’t need to flush files in exact age order – a batch of old files will do.

New implementation of flushing at CSIRO

- Part A: scanning
- Eliminate the sort (to save processing time). Assign files from the target filesystem as they are scanned to bins or buckets, based on the youngest of access and modify time.
- Define a starting time as the time of commencement of service, or (after the first flush) the age of the youngest file last flushed.
- Define 14 days (site policy) before present as our finishing time.
- Set up say 101 buckets – bucket 0 for files older than our starting time (some may have escaped!), and a linear mapping of times to buckets.
- As the scan proceeds, save just the path name into each bucket, with one bucket for files and one for directories.

Scanning for candidates for flushing

Oldest Newest

0 1 2 3 4 5 6 7 8 9 … 100

Keep these | Never need to create these …

0: <date0
1: date0–date1
2: date1–date2
3: date2–date3
4: date3–date4
…
100: date99–cut-off (e.g. 14 days)

New implementation of flushing at CSIRO

- We are never going to flush 100% of the files, or even many more than say 10%, so we need to save only perhaps 10 to 20 buckets: each will represent about 1% of the time period in question, and should hold around 1% of the files and 1% of the data
- The buckets can be saved on disc, thus obviating the need for large memory. The time boundaries need to be recorded.
- Do the scan in advance, not just when a flush is needed: overnight, or weekly or monthly, or only when a new scan is likely to be needed.

New implementation of flushing at CSIRO

- Part B: flushing
- When a flush is needed (we check every 5 minutes), the flush process looks at the bucket containing the names of the oldest files. It then reads this bucket, and for each file check that the file still exists, and that its access and modify times are not later than the bucket's upper limit: if all is well, the file can be removed, and recorded. If not, just skip the file.
- "It probably works"
- Having done one bucket, it can be moved aside (for the record), and the file system checked against the desired threshold. If more needs to be done, the bucket containing the names of the next oldest files can be dealt with in the same manner, until enough buckets have been dealt with to reach the threshold.

New implementation of flushing at CSIRO

- Finally, if the threshold has not been reached, but all the buckets have been processed, it is time for another scan. Avoid this by always ensuring that there are plenty of buckets available.
- Flushing can start promptly when needed, since a list is ready to go. There is the extra expense of a lookup for the file's existence and times to be added, but this is small compared with having to scan the entire filesystem.
- There is no need to rescan while old buckets remain: once a bucket is done for a time interval, no new entries are likely (files should always go forward in time).
- Coding, testing and implementation done in 2016, after announcements to users.

Implementation

- Working!
- *The scan time no longer matters!*
- Flushing starts promptly when triggered – a few seconds
  - With our old code, could take hours to respond.
- Latest timings

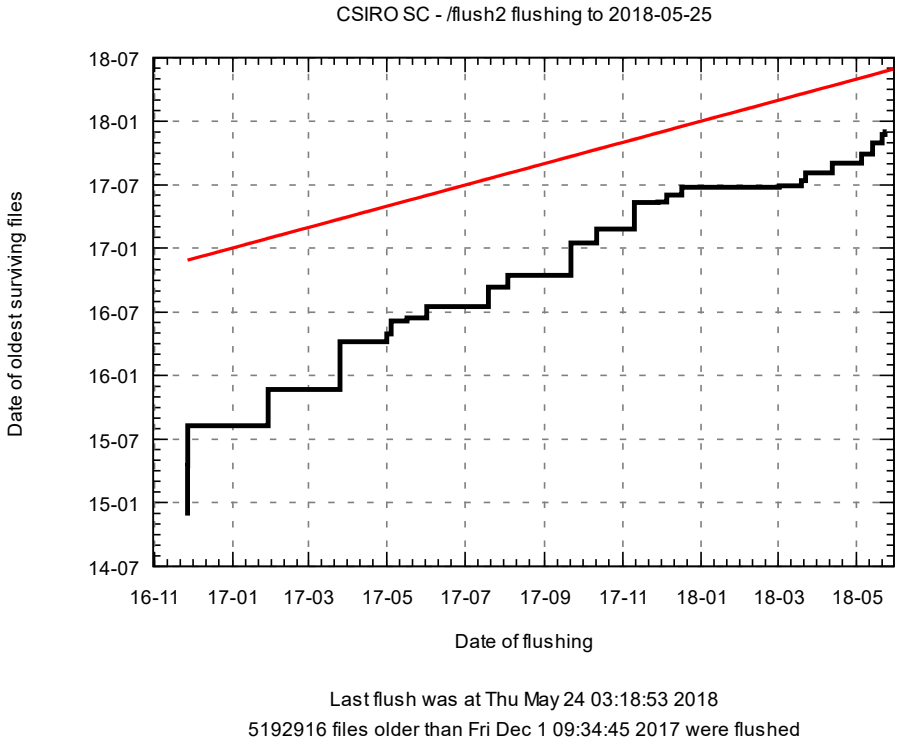
Filesystem	Files (M)	Scan time (minutes)	Flush time (minutes)	Removals (M)	Average wait (s)
/flush1	30	96	32	1.98	2
/flush2	33	26	9	4.29	7

- Buckets were nowhere near the uniformity I hoped for:
  - Saved 50 buckets instead

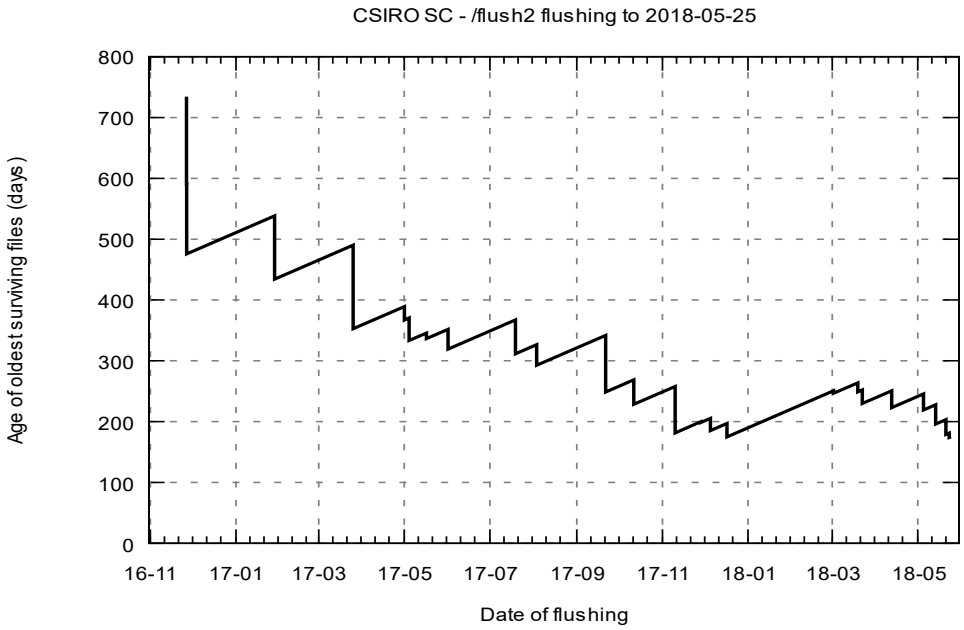
Recent work

- Production hardening
  - Responded to incident where someone accessed all the files, leaving nothing to flush
  - Now save 100 buckets spanning whole date range – not much extra work
- Time and date consistency
- Keep lists of flushed files
- Can provide lists per user of files at risk
- Open source – on bitbucket

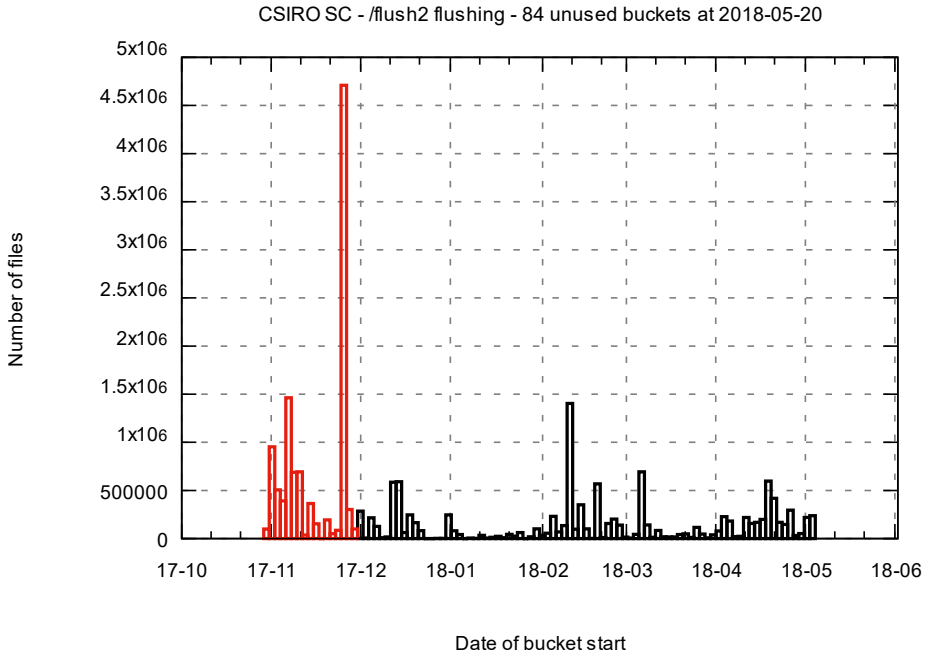
Implementation: flush dates and surviving file ages



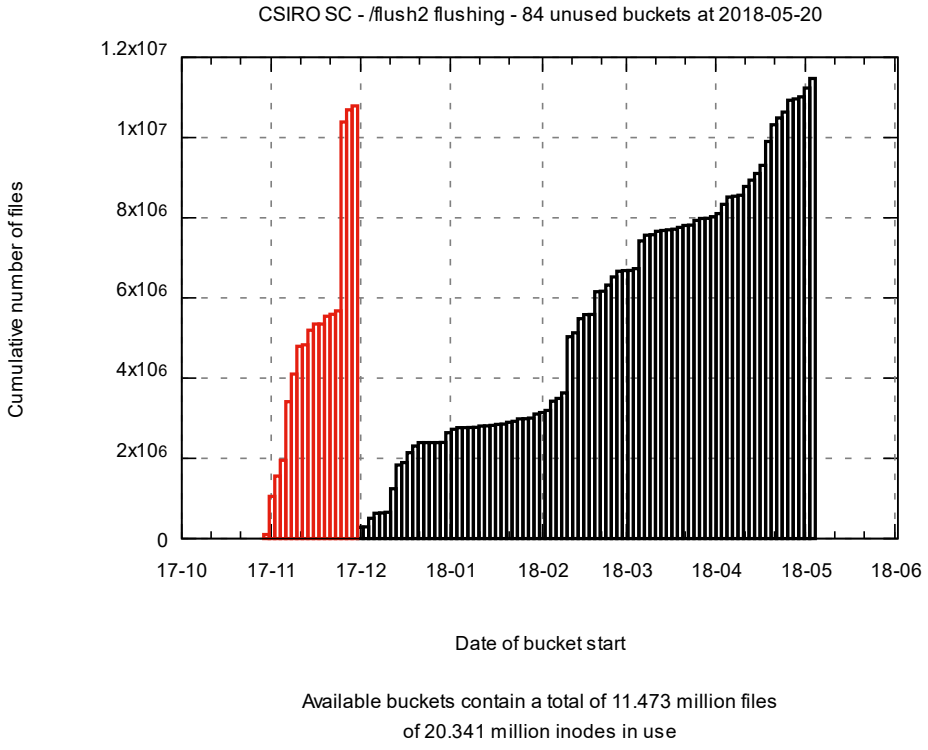
Implementation: age of surviving files



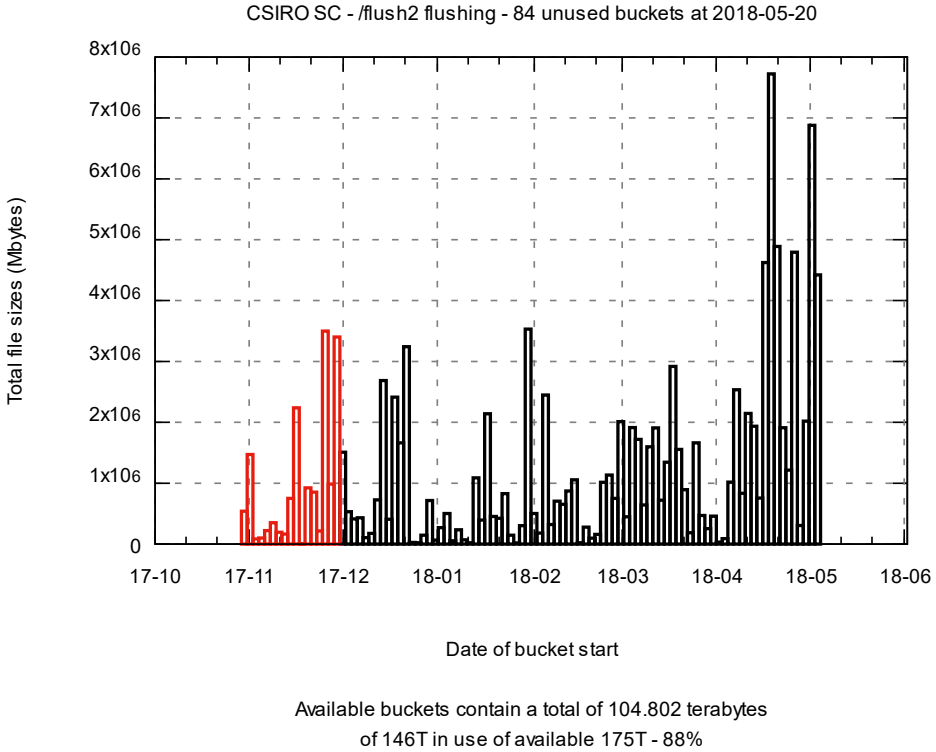
Implementation: numbers of files



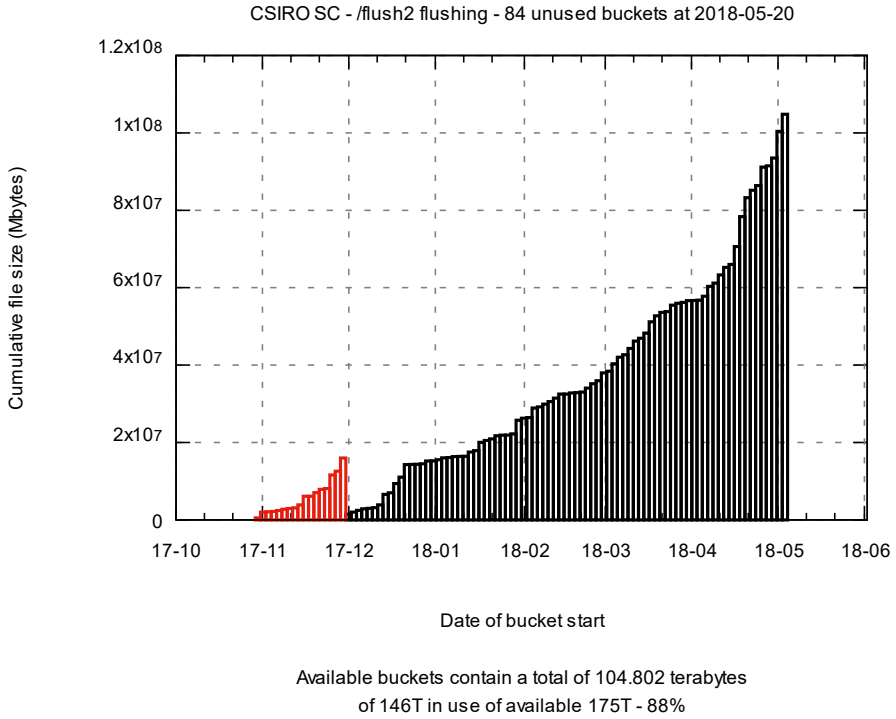
Implementation: cumulative numbers of files



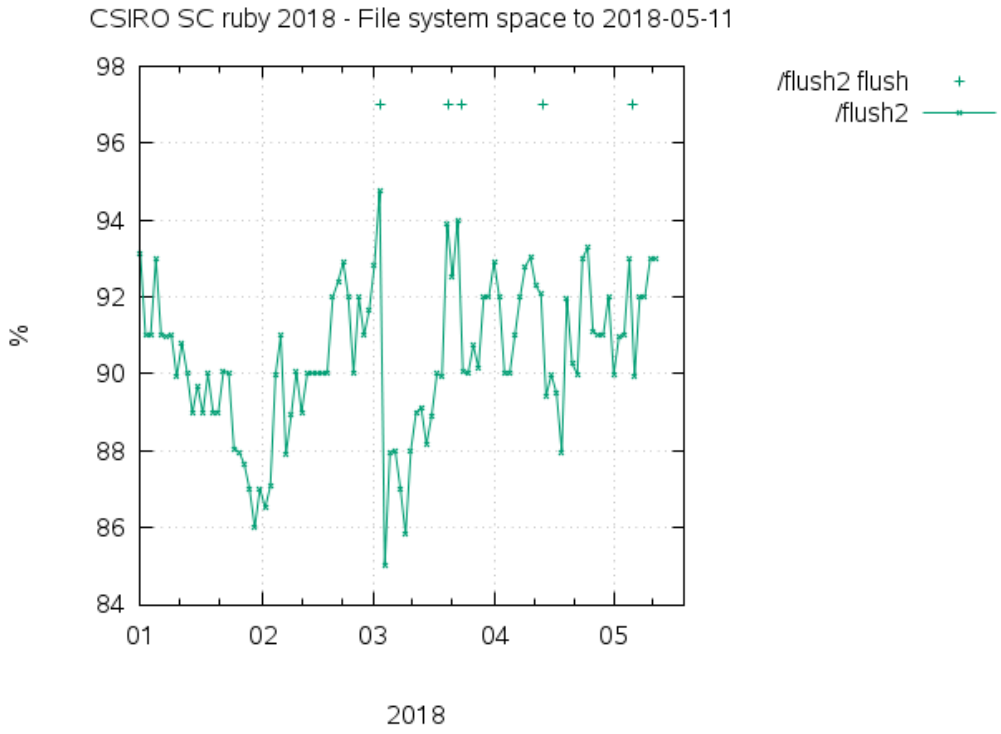
Implementation: file sizes



Implementation: cumulative file sizes



Implementation: space %



Further work

- NERSC interested in adopting this design – possible collaboration?
- Could save file sizes with pathnames, and sort buckets, so flush would start with biggest files
- Could do scanning in parallel – separate scans for each metadata server
- Could run flushing in parallel – separate flushing for each metadata server, and a separate thread for each bucket.
- Extension coded to allow flushing of part of a filesystem

Conclusion

- Policies for temporary storage
  - necessary for users, systems staff and management and productivity of users
  - range of options: maximise the value of the resources
  - need to communicate the policies (beforehand!)
- Implementing policies
  - necessary, to avoid disasters and wastage
  - tends to be over-looked
  - disasters in waiting (users' ignorance and complacency), masked by reliable hardware (mostly)
  - mustn't add to the disasters!
- New scalable flushing methodology from CSIRO

Conclusion

- With scratch under control:

  Death of {/,/g/}data!

- CSIRO looking at abolition of /data area
- Hopelessly unmanageable storage!
- Users can use either scratch, or HSM-managed, or Bowen Research Cloud
- Prototype utility to mirror scratch areas into persistent storage, and have ability to re-create after flushing
- Another talk!

Acknowledgements

- Jeroen van den Muyzenberg
- Steve McMahon
- Peter Edwards

Thank you

CSIRO IMT Scientific Computing
Robert C. Bell
CSIRO HPC National Partnerships
t +61 428 108 333
e Robert.Bell@csiro.au
w www.csiro.au