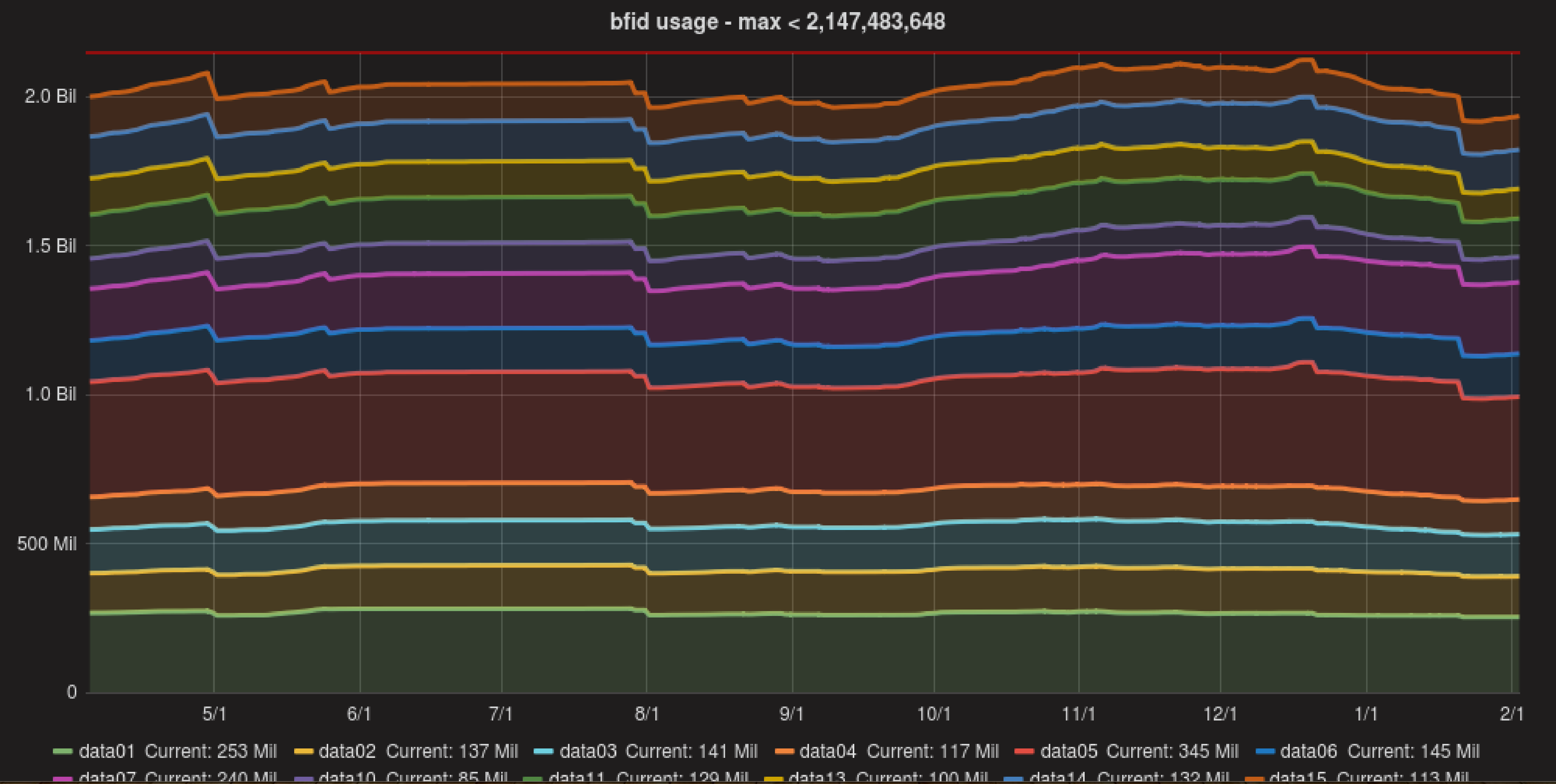
DEALING WITH > 2 BILLION FILES

WHAT HAPPENS WHEN YOU REACH THE LIMIT

- Fortunately not a lot

WHAT WE DID

- We ran dmhdelete (we'd cut over to new server and didn't copy the cron over)
- We started monitoring it. We found the quickest and least disruptive option was to run dmscanfs and count the number of bfids and then graph it.
- We deleted a relatively small number of files we'd categorised as temporary ~60 million
- We started planning what to do next

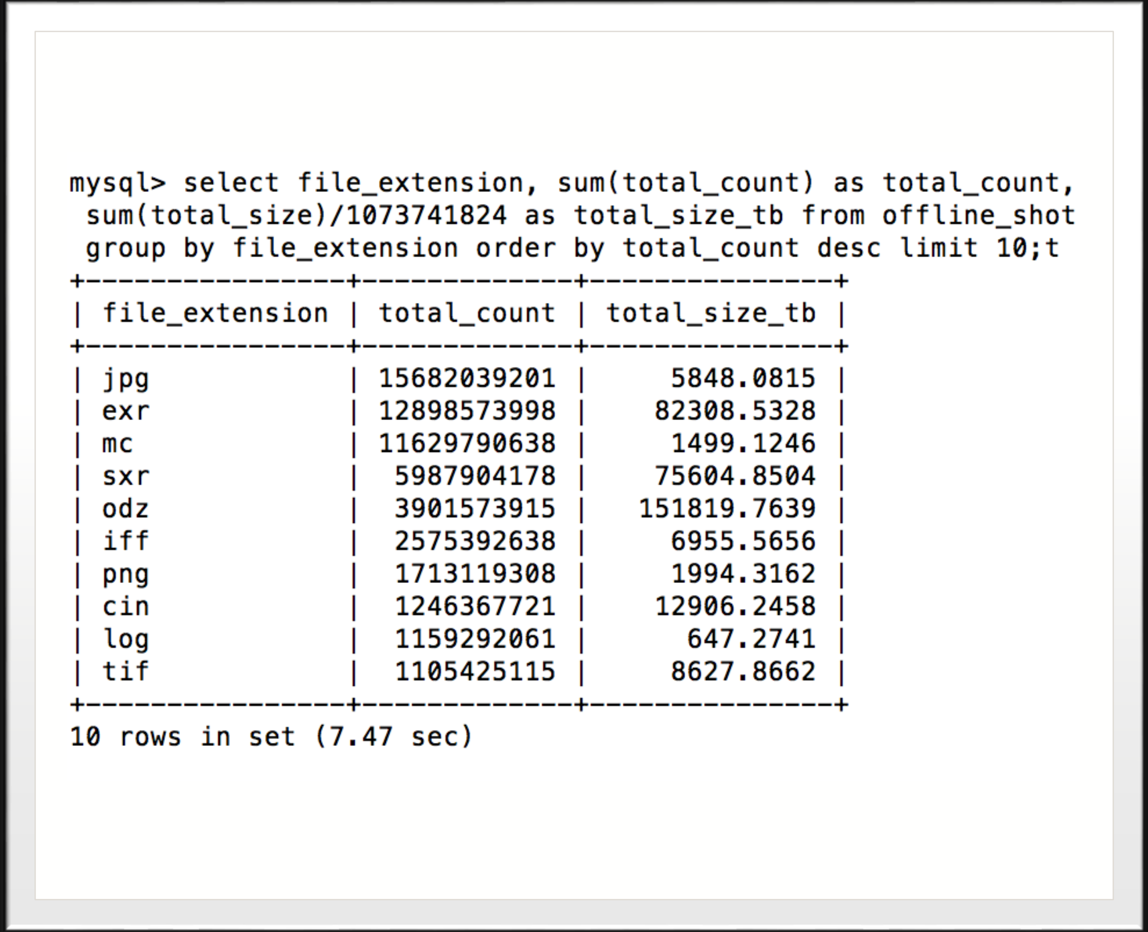

bfid usage - max < 2,147,483,648
2.0 Bil
1.5 Bil
1.0 Bil
500 Mil
0
5/1
6/1
7/1
8/1
9/1
10/1
11/1
12/1
1/1
2/1
data01 Current: 253 Mil
data02 Current: 137 Mil
data03 Current: 141 Mil
data04 Current: 117 Mil
data05 Current: 345 Mil
data06 Current: 145 Mil
data07 Current: 240 Mil
data10 Current: 85 Mil
data11 Current: 129 Mil
data13 Current: 100 Mil
data14 Current: 132 Mil
data15 Current: 113 Mil

OPTIONS DISCUSSED

- Buy more disk so we didn't need to send as much data to tape
- Setup another DMF instance
- Setup an alternative HSM that doesn't have the limitation
- Use an object store with a NAS gateway
- Use mediaflux
- DELETE

PHASE 1 – INFORMATION GATHERING

- To delete we required more information so we created a text index of all the filesystems and used some fast storage and an idle machine with lots of cores to summarise the data into file extension, size, sitetag, and 4 levels of directory structure and put it into a database. Managed to process a 500GB file in 2 hours in python and 12.5 minutes in c++ when it was later rewritten



```
mysql> select file_extension, sum(total_count) as total_count,
 sum(total_size)/1073741824 as total_size_tb from offline_shot
 group by file_extension order by total_count desc limit 10;t
+----------------+-------------+---------------+
| file_extension | total_count | total_size_tb |
+----------------+-------------+---------------+
| jpg            | 15682039201 |     5848.0815 |
| exr            | 12898573998 |    82308.5328 |
| mc             | 11629790638 |     1499.1246 |
| sxr            |  5987904178 |    75604.8504 |
| odz            |  3901573915 |   151819.7639 |
| iff            |  2575392638 |     6955.5656 |
| png            |  1713119308 |     1994.3162 |
| cin            |  1246367721 |    12906.2458 |
| log            |  1159292061 |      647.2741 |
| tif            |  1105425115 |     8627.8662 |
+----------------+-------------+---------------+
10 rows in set (7.47 sec)
```

PHASE 2 – DECISION MAKING

- We deleted ~200 million files which could be reproduced
- After 6 weeks of heated discussion we decided to:
  - Purchase 800TB of storage to buy us some time
  - Purchase infrastructure and write code to retroactively encapsulate files at a decided directory level. For us that means we decided to encapsulate pieces of work ”shots”

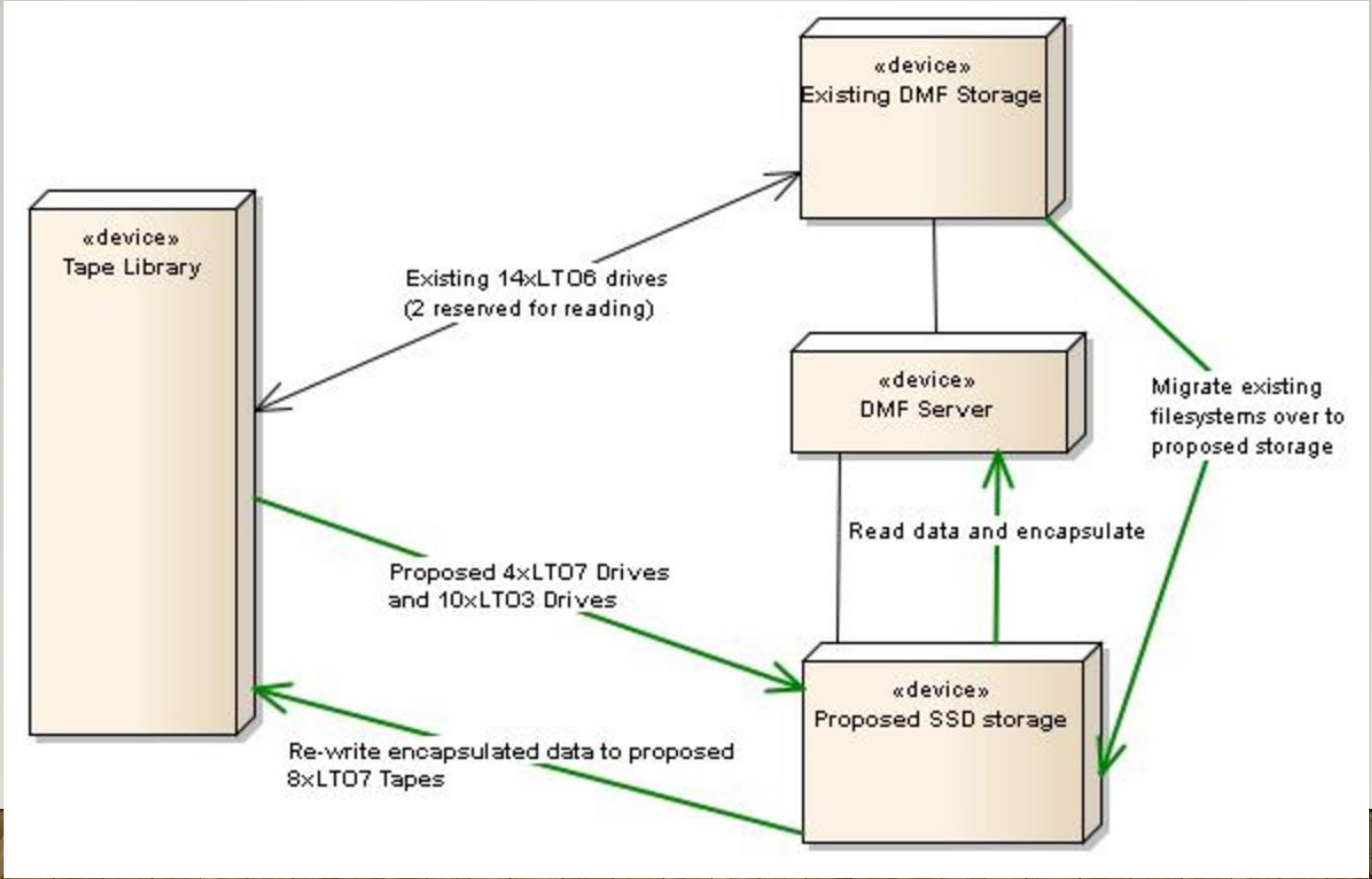
WHAT IT LOOKS LIKE

- We need to reduce the file count on DMF at the same rate or at a greater rate than we consume on average. Right now we're averaging 30T per day over the last three months and the trend suggests we're tending towards the weekly maximum of around 45T per day.

- To sustain those rates and encapsulate old shows into single files per shot we'll need the following:
  - Enough additional tape drives to sustain the same rate plus a little more so we don't lose ground. This equates to 4 drives reading and 8 drives writing, to accommodate this expansion an additional frame is required in our tape library. When reading from older tapes we need more than double the amount of tapes to hope to sustain throughput so we've requested 10 more LTO3 drives which are relatively inexpensive.
  - A storage array with enough performance to sustain the writing from tape > reading to encapsulate > writing to encapsulate > reading to tape drives again
  - A memory upgrade for the DMF server as encapsulation requires cataloguing all the file information in memory before writing them out
  - High throughput fiber channel cards to accommodate the modern storage array

ENCAPSULATION FORMAT - DAR

- Open source
- Ability to have an external catalogue file, which provides byte offsets for file data within archive files, which can then be used for partial file recalls by DMF before file extraction
- Retains file information like atime, mtime, owner and extended attributes (If you want)
- Supports hard links and symbolic links
- Has a track record of long term support
- Has an API (c++)
- Reading the archive data files not required to find where data is for restore

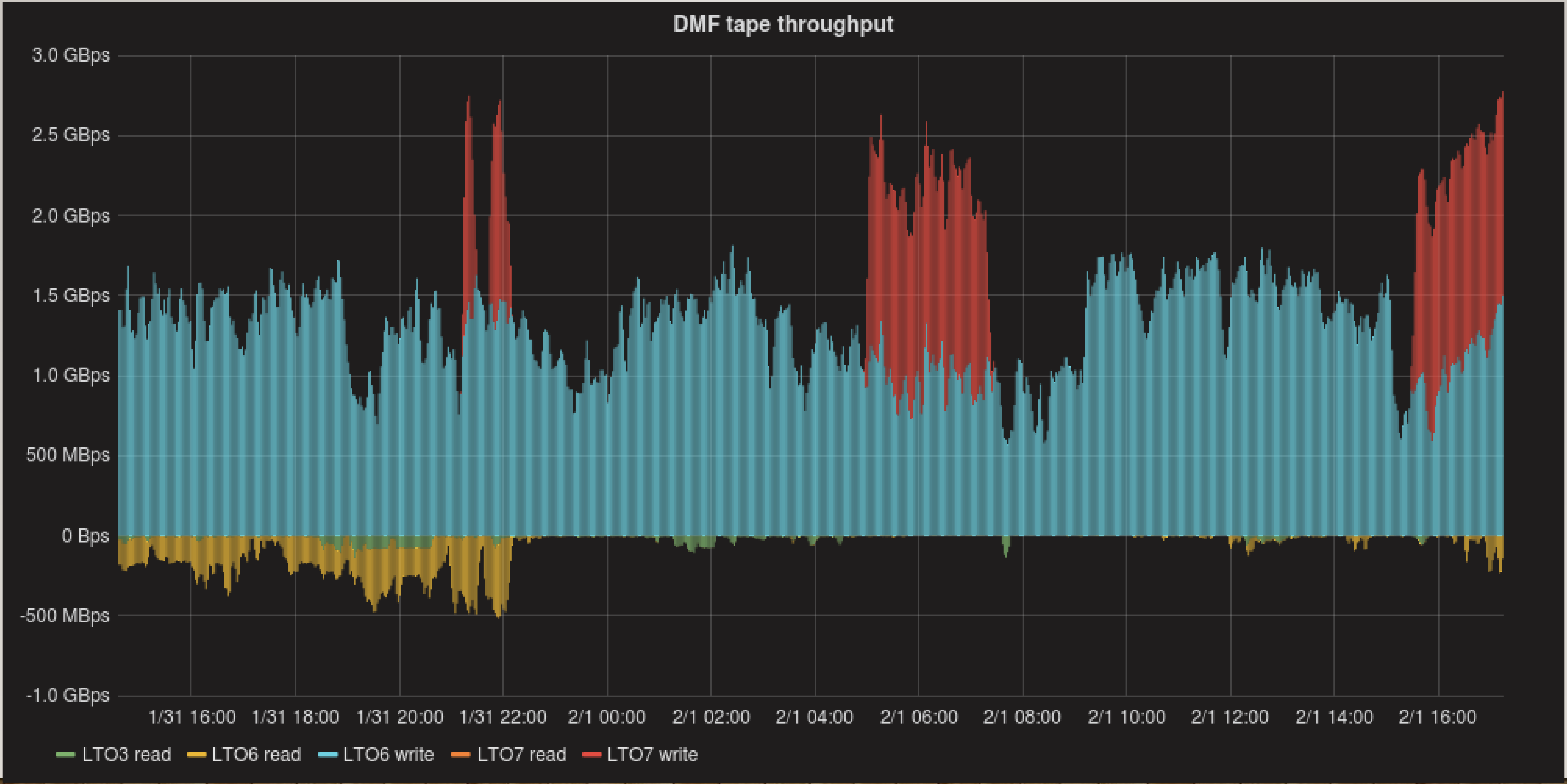

«device»
Existing DMF Storage
«device»
Tape Library
Existing 14xLTO6 drives
(2 reserved for reading)
«device»
DMF Server
Migrate existing
filesystems over to
proposed storage
Read data and encapsulate
Proposed 4xLTO7 Drives
and 10xLTO3 Drives
«device»
Proposed SSD storage
Re-write encapsulated data to proposed
8xLTO7 Tapes

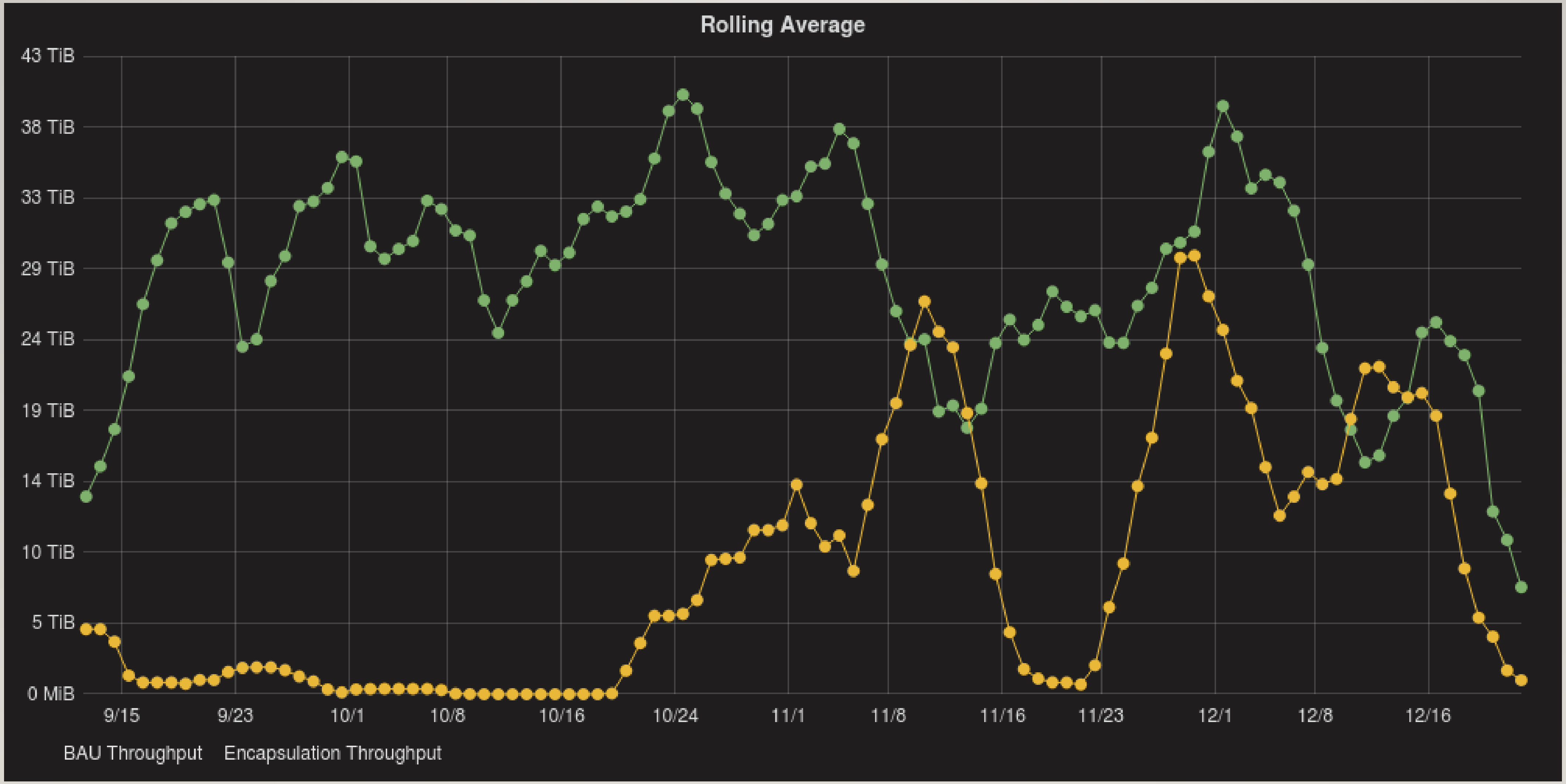
OPTIMISATIONS MADE

- Built a shot to tape mapping so we can order encapsulation based on highest gains per tape physically loaded into the library
- Also use tape drives from the BAU pool when free
- Automate exports and imports based on what shots are going to give the best gains
- Used copy and archive and pipelined requests asynchronously
- Modified dar CRC algorithm and default chunk size to reduce CPU overhead from 90%+ of a single CPU to ~10%
- Order recalls by position on tape so we can limit ourselves to 10,000 outstanding dmcopies per tape and still avoid tape rewinding
- Process all shots where all required tapes are in the library that will fit into the SSD filesystem
- Created a disk MSP for the catalog files so if they're removed from disk by accident they come back quickly
- Made the encapsulation process autonomous

RESULTS

- Encapsulation at chosen directory level reduces file count by a factor of 40,000
- So far have encapsulated 1900TiB and 275,000,000 BFIDs worth
- We’re automatically merging old tapes
- load average and CPU use not noticeably affected when encapsulation running


DMF tape throughput
3.0 GBps
2.5 GBps
2.0 GBps
1.5 GBps
1.0 GBps
500 MBps
0 Bps
-500 MBps
-1.0 GBps
1/31 16:00
1/31 18:00
1/31 20:00
1/31 22:00
2/1 00:00
2/1 02:00
2/1 04:00
2/1 06:00
2/1 08:00
2/1 10:00
2/1 12:00
2/1 14:00
2/1 16:00
LTO3 read
LTO6 read
LTO6 write
LTO7 read
LTO7 write


Rolling Average
43 TiB
38 TiB
33 TiB
29 TiB
24 TiB
19 TiB
14 TiB
10 TiB
5 TiB
0 MiB
9/15
9/23
10/1
10/8
10/16
10/24
11/1
11/8
11/16
11/23
12/1
12/8
12/16
BAU Throughput
Encapsulation Throughput

OTHER ADVANTAGES

- We’ve got a simpler structure so if we decide to move off of DMF or supplement it in any way (optical media maybe) we can
- Housekeeping tasks will get better over time
- Recall throughput for an entire shot is improved
- Having easy access to chunknumber on tape also means we can improve individual file recall performance