Studying Scientific Data Lifecycle in On-demand Distributed Storage Caches

Julian Bellavita, Alex Sim, Kesheng Wu at LBNL
Inder Monga, Chin Guok at ESnet
Frank Wurthwein, Diego Davila at UCSD

Introduction

- Background – HEP community and big data
- The XrootD system
  - Stores, distributes, and caches large datasets for HEP community
  - Storage cache deployed by ESnet: 1 Xcache node in Sunnyvale, CA
    - Cache size ~= 40TB
- Data Source of Our Study: XrootD server logs
  - Information about file operations could be identified by keywords correspond to these operations
  - Server logs captured detailed file access information for us to study data access patterns
- Brief outline of the study
  - File read operations
  - File lifetimes
  - Cache simulator

General Methodology

- Primary programming tool – Python on the NERSC Jupyter Hub
- General approach
  - 1. Identify keywords/keyphrases corresponding to certain operation (e.g., open, read, vector read, cache miss, etc.)
  - 2. For each XrootD file in the specified date range, parse each line
    - If the line has the keyword, extract relevant information (time stamp, read size, etc)
  - 3. Store information in a data structure, typically hashmap or list
  - 4. Compute statistics

File Reads

- Two kinds of read operations
  - 1. Simple read operations – ‘**req=read**’
  - 2. Vector Read (readv) operations - reads several blocks starting from a specified offset – ‘**fh=0 readV**’
- Design issue – readv operations don’t specify the corresponding file in the server line
  - Solution – search for open requests **‘open rat’ or ‘open r’. These include the user ID and the job ID, as well as filename**
  - **Read operations include the same user ID and job ID – mapping user ID+job ID to filename lets us tie read operations to their file**
- Read operations also specify their sizes and offsets – taking the form NNNN@MMMM
  - NNNN -> size of read request
  - MMMM -> offset of read request

File Reads

- First thing we studied – number of read operations issued towards a given file in a month
- Originally used monthly means to visualize mean read operations.
- This did not work well
- We replaced the mean plot with these histograms

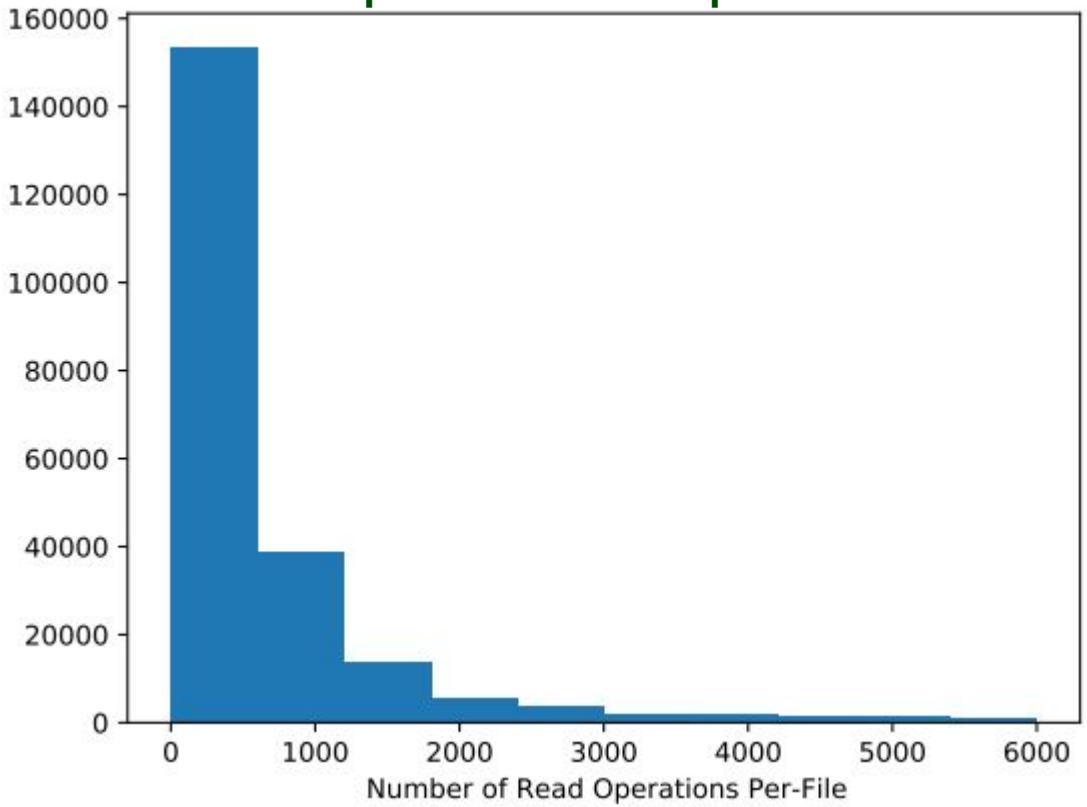


Figure 1 (a): Distribution of Total Read Operations Per-File for Jan 2021-Sep 2021

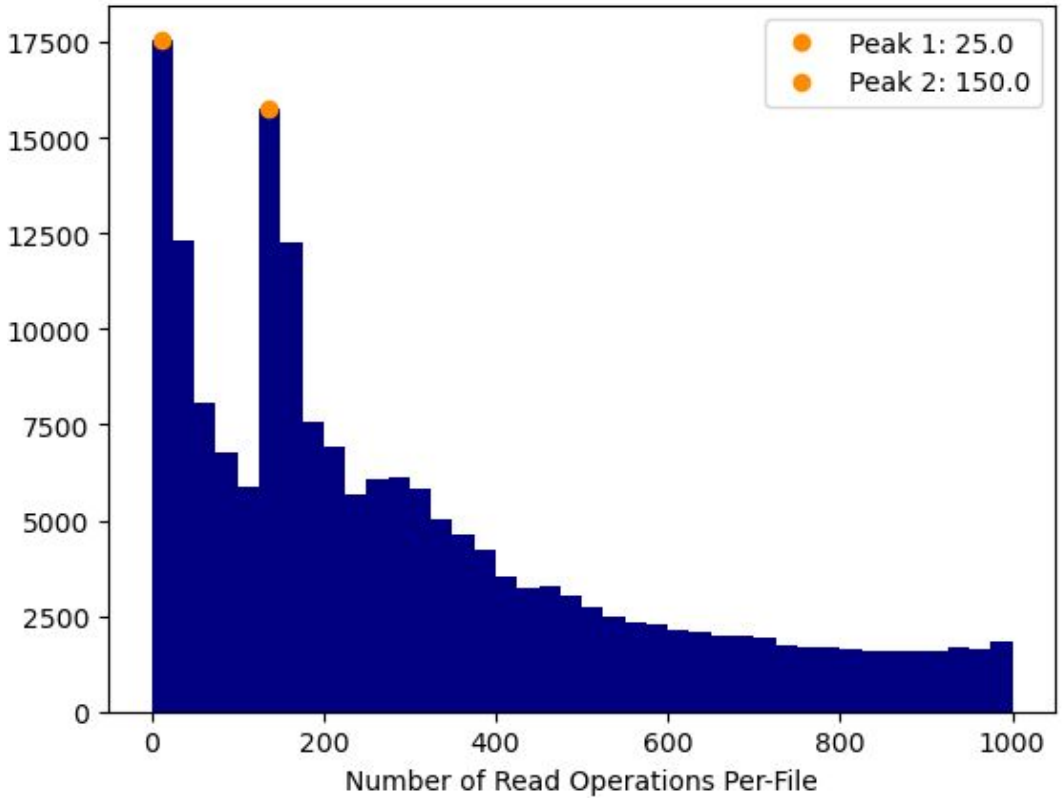


Figure 1 (b): Zoomed-in, finer-grained distribution of Total Read Operations Per-File for Jan 2021-Sep 2021

File Reads

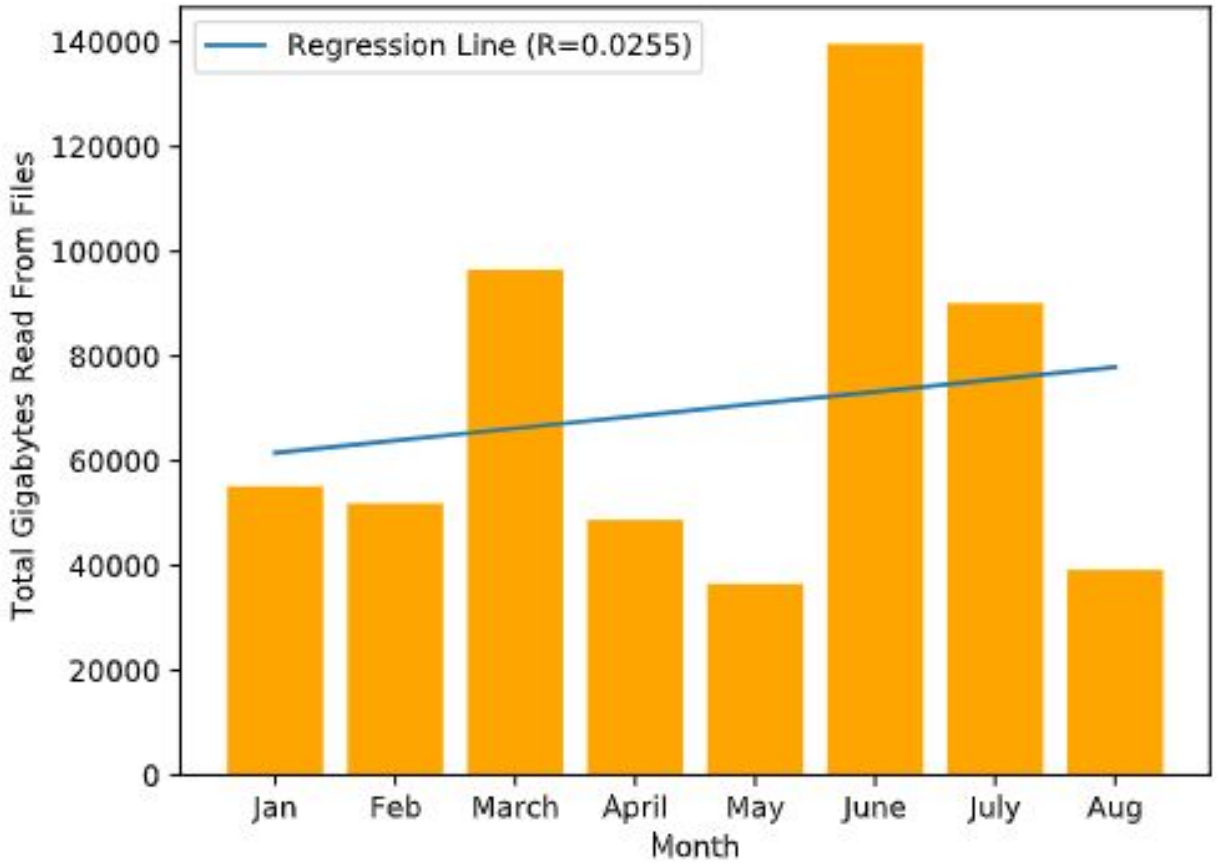


Figure 2: Monthly total size of file reads for Jan 2021-Aug 2021.

File Reads

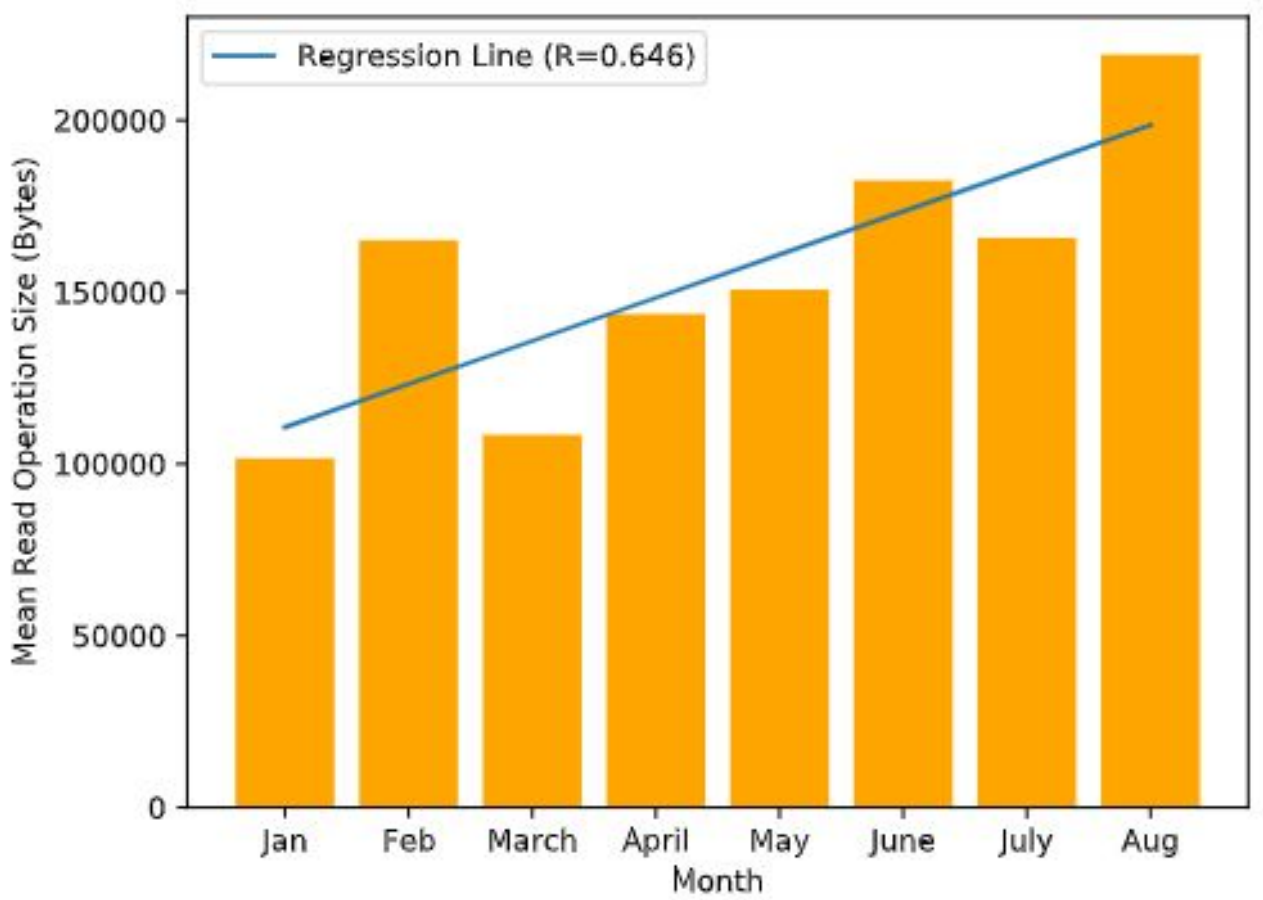


Figure 3: Mean size of file read operations for Jan 2021-Aug 2021. Global mean = 154,632B

File Reads

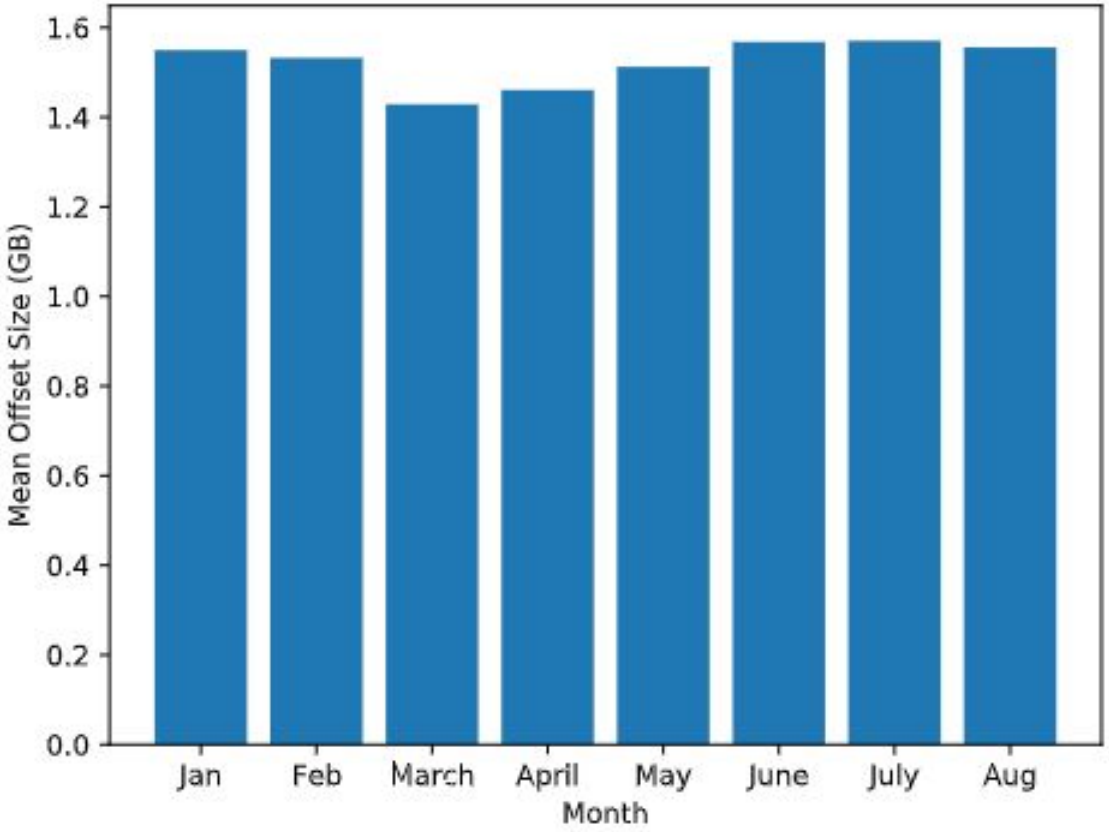


Figure 4: Mean offset size for read operations from Jan 2021- Aug 2021.
Global mean = 1.52GB

File Lifetimes

- We want to know how long files tend to remain open and in use
  - Primary benefit — improved caching policy [1]
- Measure the time between open requests and close requests
  - open -> ‘open r’ or ‘open rat’
  - close -> ‘prefetch score’
  - We use a dictionary mapping filenames to tuples of the form (s, e), where s-> beginning timestamp and e-> end timestamp
  - Enables the computation of statistics regarding file lifetimes
- If another open request is issued after a close request, but before a certain cutoff point (1.2 days), its treated as a continuation of the lifetime

[1] Luis Thomas, Sebastien Gougeaud, Stéphane Rubini, Philippe Deniel, and Jalil Boukhobza. 2021. Predicting File Lifetimes for Data Placement in Multi-Tiered Storage Systems for HPC. SIGOPS Oper. Syst. Rev. 55, 1 (jun 2021), 99–107

File Lifetimes

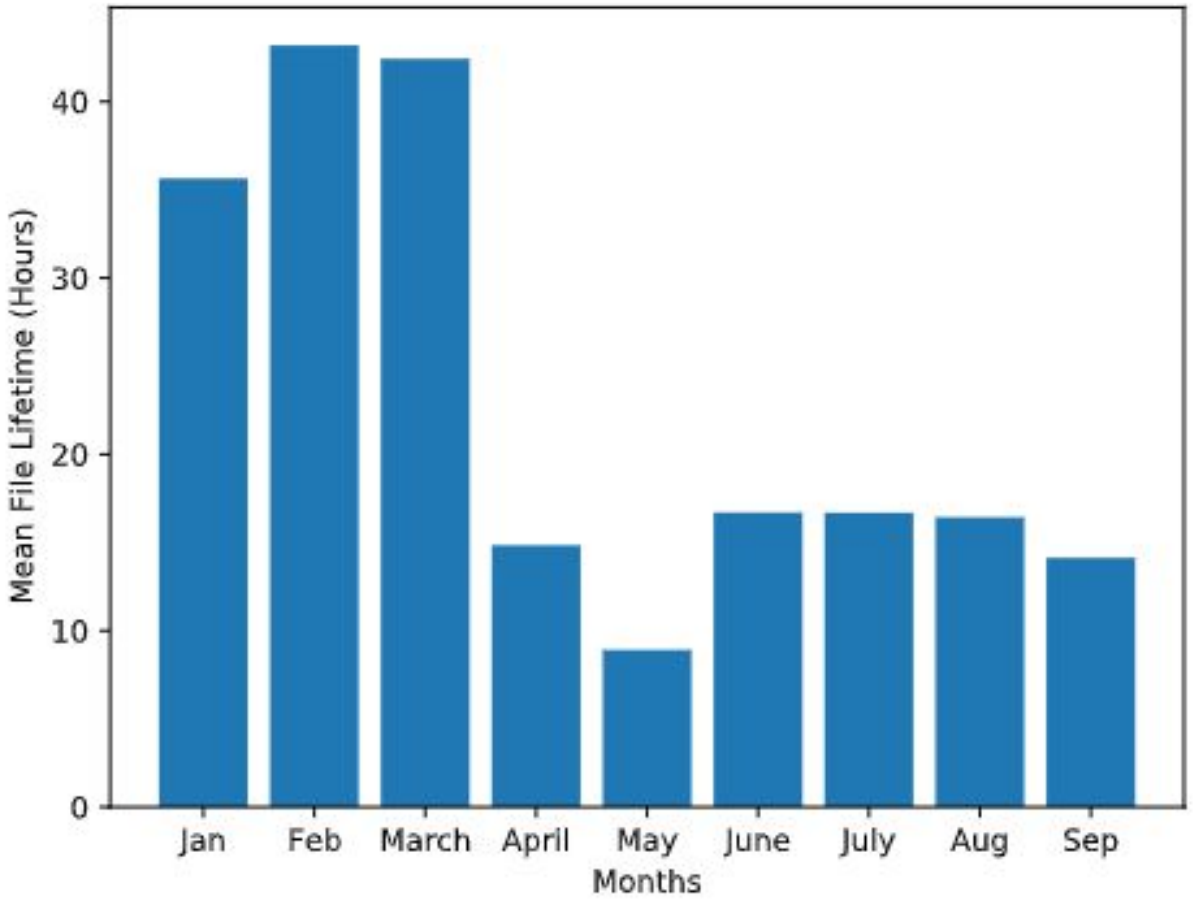


Figure 6(a): from Jan 2021-Sep 2021 using a threshold of 1.2 days. Global mean = 23.23 hours.

File Lifetimes

Using monthly means to visualize the data… not the most informative

- **Didn't reveal anything interesting**
- **Misleading global mean**

We now include two histograms to visualize the data… much better

Figure 6(b): Distribution of file lifetimes for Jan 2021-Sep 2021

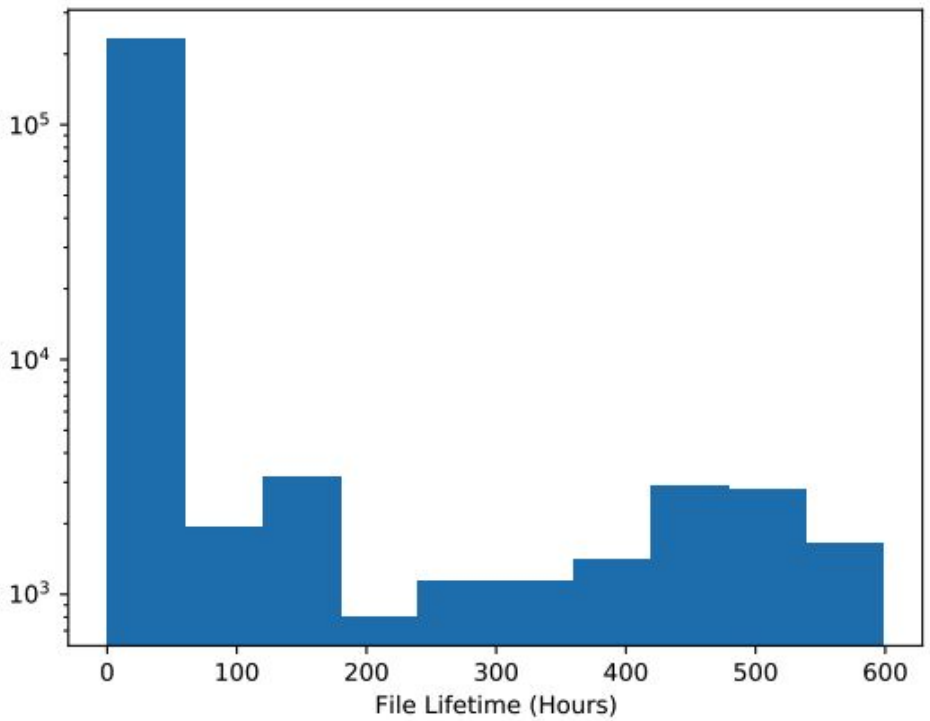
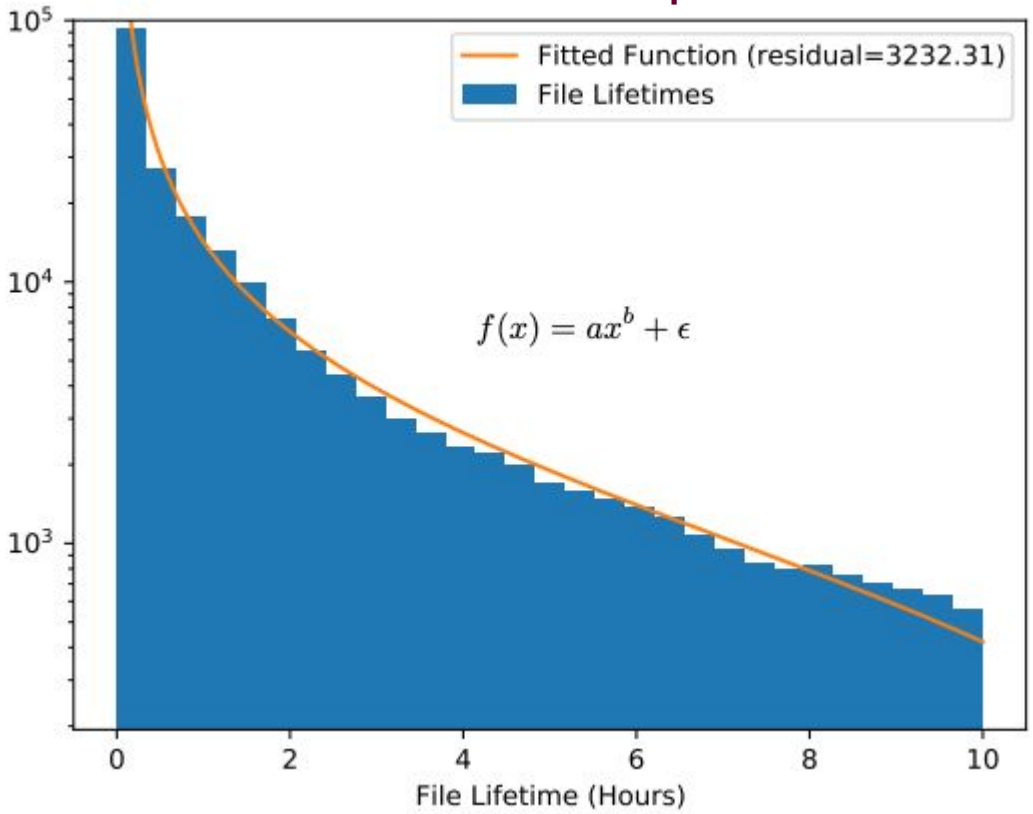


Figure 6(c): Zoomed-in, finer-grained distribution of file lifetimes for Jan 2021-Sep 2021



a	b	ϵ
15227.387	-1.031	-995.488

Table 1: File Lifetimes Fit Function Parameters

< 1 Hour	< 5 Hours	< 10 Hours
54.6%	78%	83.8%

Table 2: File Lifetime Percentages

Cache Simulator

- We need 2 things from the server logs to simulate the cache
  - 1. Cache misses (file, size)
  - 2. Read operations
- The second thing is provided by already described procedures
- Now, for the first…
  - cache misses are denoted by **'successfuly read size from info file = NNNN'**
    - Misspelling is intentional
    - NNNN -> transfer size
    - Also includes file name elsewhere in the line
- Cache Simulator has 2 modes

Cache Simulator

- **Data structure**
  - We use an Ordered Dictionary that maps filenames to a File class containing various metadata, e.g., size, access timestamp
  - Can specify a cache size
- **Mode 1 – Compute hit rate given cache size**
  - When a cache miss is found…
    - Add file to the front of the dictionary
  - If the total size of files in the cache exceeds the cache size
    - Evict last element from the ordered dictionary (cache replacement policy Least-Recently Used, LRU)
  - When a read operation is found
    - Check the filename it's issued towards (using the same procedure as before)
    - If that file is in the cache, move it to the front, count one hit
  - In either case, count one access
  - At the end, divide hits/accesses — that's the hit rate

Cache Simulator: Cache Size Significantly Affects Hit Rate

Hit rate scales with the cache size until about 54TB

On the left – total amount of data transferred for August '21 is ~60TB

Close to observed hit rate

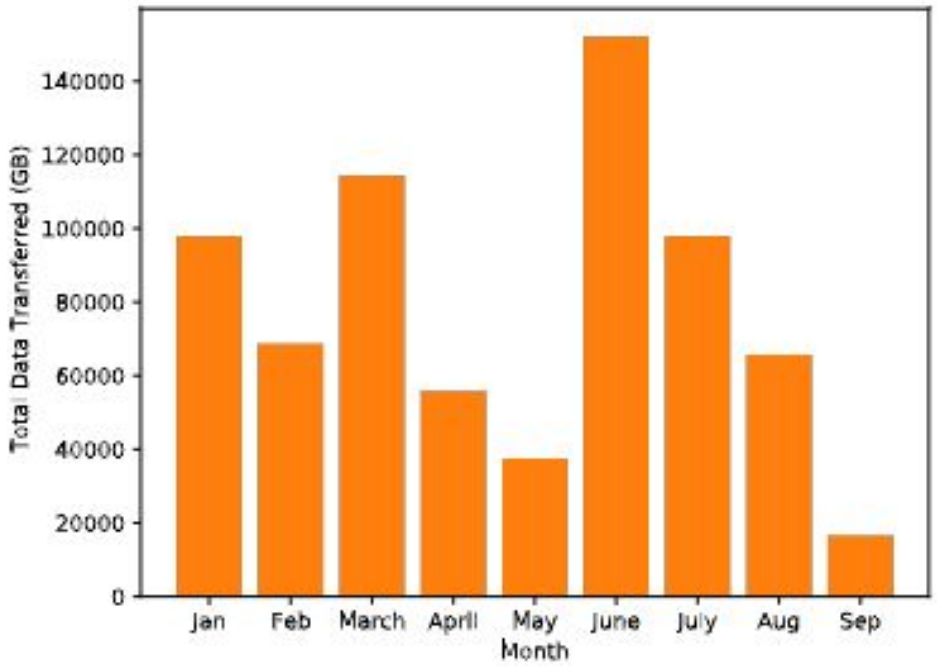


Figure 7: Total amount of data transferred to the cache for Jan 2021-Sep 2021

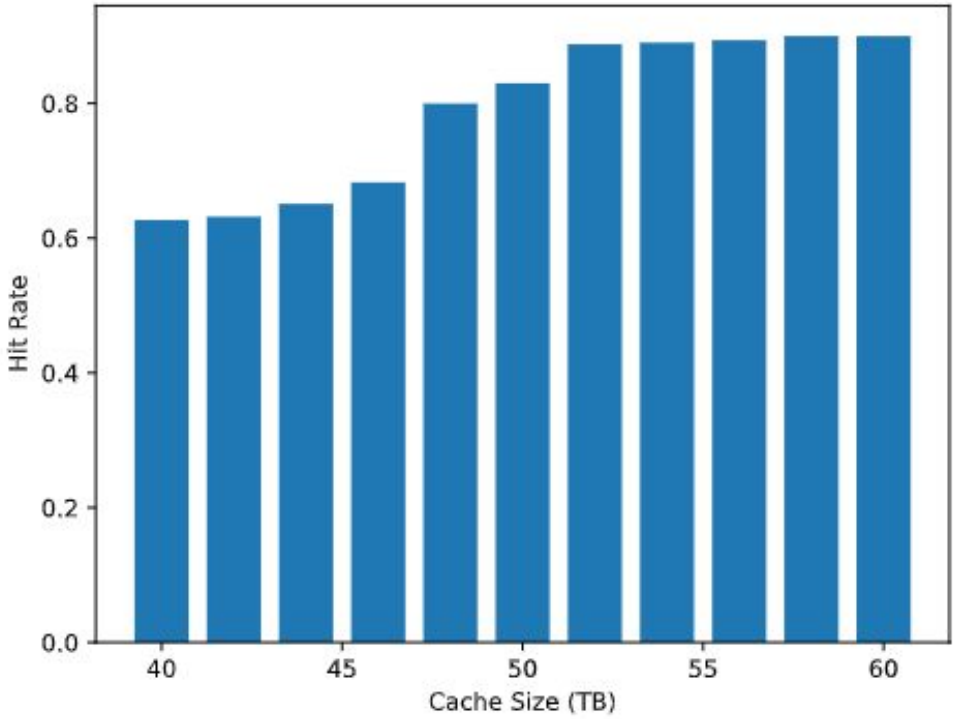


Figure 8: Cache hit rates for Aug 2021 for a range of cache sizes (40TB-60TB)

Cache Simulator

- Mode 2 – Time to fill up cache

- Same general procedure, but stop once the cache fills up and return the difference between the starting timestamp of the analysis and the timestamp of the last access

Cache Simulator

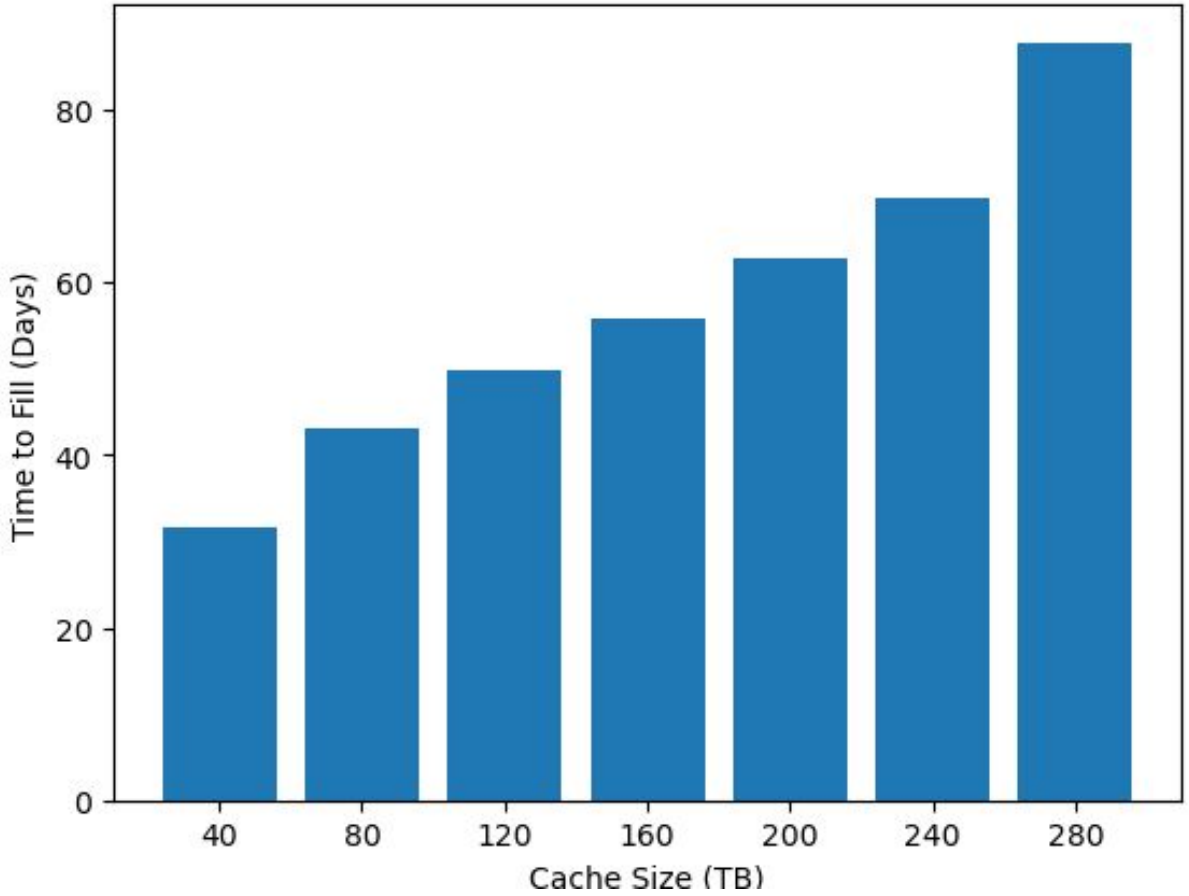


Figure 10: Time it takes to fill caches of sizes 40TB-280TB

Future work

- Expand cache simulator to model different eviction policies (e.g. random, lifetime-based)
- Expand cache simulator to be able to simulate different access rates
- Develop machine learning models that can predict file lifetime lengths

Conclusion

- We provide summary statistics regarding…
  - 1. Numbers of file read operations
  - 2. Sizes of read operations
  - 3. File lifetimes
- We simulate cache behavior to show the following
  - 1. How hit rate changes as cache size increases
  - 2. How cache contents change as a function of time and hit rate
  - 3. How long it takes to fill up various cache sizes
- These insights can inform hardware and protocol decisions for future XCache nodes