Robust Benchmarking for Archival Storage Tiers

DongJin Lee Michael O'Sullivan Cameron Walker

Department of Engineering Science
The University of Auckland
Auckland, New Zealand
{dongjin.lee, michael.osullivan, cameron.walker}
@auckland.ac.nz

Monique MacKenzie

Department of Statistics
The University of St Andrews
St Andrews, United Kingdom
monique@mcs.st-and.ac.uk

ABSTRACT

Until recently archival storage tiers have consisted of tape-based devices with a large storage capacity, but limited I/O performance for data retrieval. However, the growing capacity and shrinking cost of disk-based devices means that disk-based systems are now a realistic option for enterprise archival storage tiers. Given the increasingly diverse options for archival storage, robust benchmarking of possible technologies for archival storage tiers is vital for reducing risk before deployment.

This paper investigates benchmarks that utilize archival workloads developed from an analysis of historical file size distributions. These benchmarks not only provide more appropriate measurements of system performance *as an archive* than traditional approaches, but we also incorporate the variation observed in the historical data to provide "best" and "worst" case workloads for benchmarking. By considering not only the usual workload, but also workloads at either end of the archival workload spectrum, our benchmarking is robust, i.e., it provides measures of performance for the envelope of typical archival workload observed from empirical data.

1. INTRODUCTION

The archival tier or tiers play a critical role in data management as all current data will eventually be archived. Typically tape-based systems have been used for archival tiers, but the use of disk-based storage – hard disk drive (HDD) – for archival storage tiers is growing in popularity.

Adams et al. [5] argue in favor of disk-based storage by making three observations: 1) retrieval patterns for data in archival storage are similar to retrieval patterns from online storage, so archival storage should have similar random read performance to online storage; 2) tape-based storage performs best in a write-once read-*never*/*maybe* archival situation (also known as a "pure" archive), but reasonable amount of archival data is now retrieved eventually (in fact, the observations show that 50–75% of archival data received updates, i.e., was read and then written again); and 3) files that are infrequently accessed can still be the target of software such as Google crawl or integrity checking. They also noted that the large-scale retrieval of archival data is dealt with much better by disk-based storage given the sequential-access behavior of tape-based devices.

Since it is now reasonable to expect medium to large-scale retrieval of archived data, but with a long time between retrievals of a particular file, it makes sense to use two tiers for archival storage. A disk-based tier for recently archived data and a tape-based tier for storing data that is "truly historical" and will probably never be accessed again.

When designing storage systems in practice, experienced storage architects carefully consider a range of alternatives from different vendors to find the technology that best meets their requirements. However, this approach relies heavily on the experience of the architect, including their ability to keep up to date with the constantly evolving storage systems technology. Often, overprovisioning is used to account for uncertainty in the storage system's workload.

In this paper we not only generate a typical archival workload by analyzing historical measurements, but we also take into account the variation in these measurements to provide an "envelope" of typical workloads. This envelope provides a robust benchmark for archival storage as it takes into account variation in typical workloads and gives not only the most likely workload, but also "lower" and "upper" workloads for more thorough benchmarking.

The structure of the paper is as follows:

Typical workload characterization – we determine the workload for an archival storage tier by the size of files in the archive. We developed a model for the distribution of file sizes in a typical storage archive using measurements from large HPC archival sites [8]. We also construct an envelope for the file size distribution using variation in those measurements. We describe the development of the model and envelope in Section 2.

Archival workload benchmark – using our envelope for the typical file size distribution, we benchmarked two storage systems: 1) the ext4 filesystem deployed locally on a RAID array; and 2) the distributed, object-based filesystem Ceph [16] with the RAID array used as an Object Storage Device (OSD). We measured the performance of these systems using several different benchmarks, including both traditional benchmarks and benchmarks built from our file size distribution envelope. We describe the measurement and experimentation in Section 3. Finally, Section 4 concludes the paper.

PDSW'11, November 13, 2011, Seattle, Washington, USA.

2. ARCHIVAL STORAGE WORKLOAD

2.1 Related work

Miller and Katz [12] observe that a tape-based tier within a high volume data storage system received a different amount of read/write requests during different time of the day (peak/off-peak time). However, their observed utilizations (applications dealing with large data sets) is very different from typical archival storage for a large institution such as a university. They also observe that tape-based archival performance is highly dependent on the access time to the first byte (in a file) and the access locality of files. Hence, to build a good benchmark for a particular system, the temporal access pattern for files in the system needs to be embedded in the benchmark.

Other approaches [3, 17] build "exact" benchmarks for an existing system by intercepting the aggregated I/Os (e.g., Logical Block Addressing – LBA) received at the disk layer and then replicating these block I/O patterns in the benchmark. However, these kinds of low-level traces cannot easily be scaled as they do not reveal actual file sizes. Hence they are good for accurately reproducing the block-level I/O of an existing archive, but not for the design of new systems.

Trager et al. [15] investigate various benchmarking approaches for filesystems and storage, and one of their findings is that a good understanding of how the filesystem/storage is going to be used is the best way to develop a good benchmarking approach. This conclusion also indicates that understanding the access pattern for a storage system is the best way to build benchmarks for that system.

In this work, we are interested in building a benchmark for a typical archival storage tier, but only for its active performance. The tier is used to store a subset of files on the online storage tiers that is identified as being inactive for "long enough" to be archived. Once archived, some of these files may be retrieved and it is the request for multiple files at once that is of particular concern (e.g., the retrieval of the directories for an employee who has been absent for an extended period of time). Hence, we are interested in the performance of the archive during archiving (when "old" files are migrated from the online storage to the archive) and multiple retrievals (when multiple files, which may not have been archived at the same time, are retrieved). Furthermore, we want to provide a robust benchmark, i.e., one that will not only test typical behavior, but also that accounts for variation in typical behavior.

There is no previous work that provides multiple samples of temporal access patterns from typical archival storage tiers. However, Park et al. [13] observe that critical factors affecting the time required to access data storage are the number of files and the size of each file. Given the lack of temporal access pattern information, we instead use available information on the file size distribution of archival storage tiers [8] to build our benchmark. Archiving works by periodically selecting files (using selection policies such as by last-accessed time) and aggregating them into an ordered batch. These batches are stored in the archive volume using a single *sequential write* (known as data migration). Given that the archival storage tier is an aggregation of these sequential writes, we assume that the file size distribution of a single sequential write is a proportion of the tier's file size distribution. We realize that this assumption is simplistic, but without temporal access pattern information, it seems reasonable. Emulating of the retrieval of multiple files from the archival storage tier is more problematic. It is likely that many of the files from a single directory will have a similar history and may be archived at the same time, i.e., be within the same sequential write. However, without temporal access pattern information, there is no way to infer which files may be retrieved in the same request. We assume that any file is equally likely to be retrieved (amongst the files in the storage) and benchmark the retrieval performance by randomly retrieving files from the archival storage tier, referred to as *random read*. This assumption is also simplistic, but, given the available information, also seems reasonable.

Both the sequential write and random read operations depend on the file size distribution of the archival storage tier. In Section 2.2 we use existing information about typical archival file size distributions to build an accurate model of a typical archival file size distribution. By incorporating variation in the typical file size distribution, we also ensure our resulting sequential write and random read benchmarks are robust. Studies on file size distributions and how they varied over time have been performed in the past. Understanding file size distributions is, independently, a useful field of research, see [7, 10, 11] for examples.

2.2 Characterizing the file size distribution

Dayal [8] provides empirical data from large HPC sites. Both archival and non-archival file size distribution are shown in Figure 1(a) using two functions: Cumulative Distribution Function (CDF) $F(x) = \Pr(X < x_i)$ and Complementary Cumulative Distribution Function (CCDF) $\bar{F}(x) = \Pr(X > x_i) = 1 - F(x)$. While the CDF shows the distribution, the CCDF shows the tail distribution, i.e., how the the CDF "tails off" to 1. The CCDF shows the distribution of large files, because it is important to make sure our workload produces a realistic number of files with large sizes. This is very important when testing I/O performance (see Section 3).

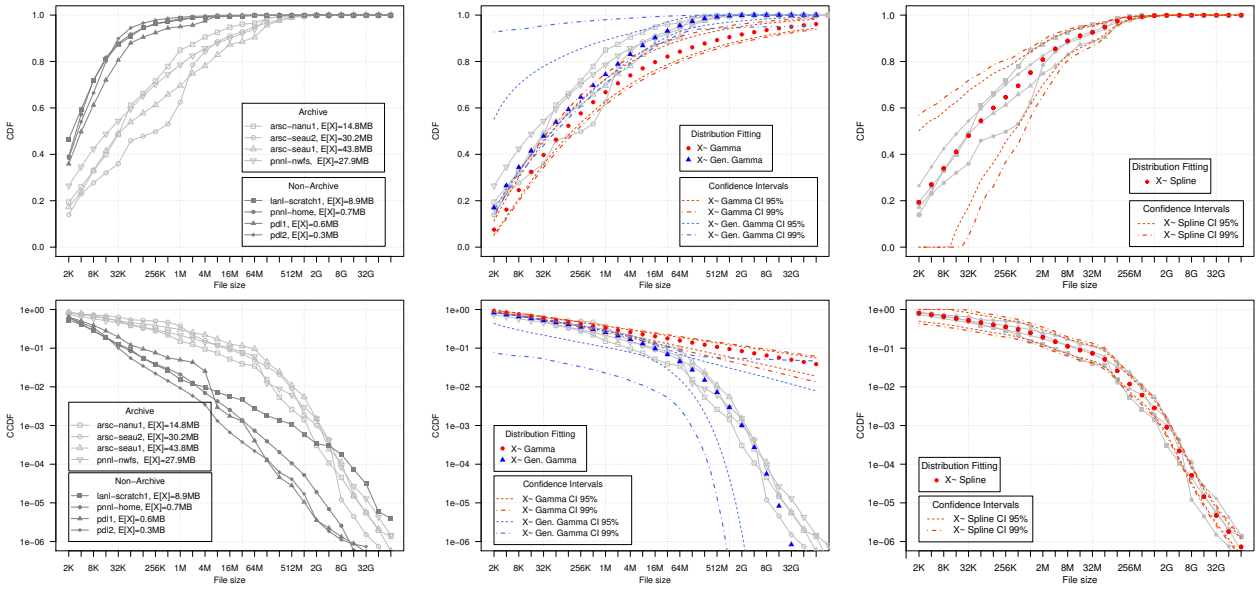
Figure 1(a) shows that there are a higher proportion of large files in archival storage than in non-archival storage. In non-archival storage 61% of files are <8KB and 81% are <32KB, while only 28% are <8KB and 36% are <32KB for the archival storage[1]. One reason why archival systems have more files and a larger volume of files is because both current and older files eventually get moved to archival storage and remain there for a long period of time. Another possible reason is the existence of techniques for reducing the overhead of many of small files by aggregating them into larger chunks [11].

To produce a model of the typical file size distribution we first tried the Gamma and Generalized Gamma distributions. The Probability Density Function (PDF) for the Generalized Gamma distribution is [14]:

$$f(x;\theta,k,p) = \frac{(p/\theta^k)x^{k-1}e^{-(x/\theta)^p}}{\Gamma(k/p)}, \text{for } x \geq 0, \text{and } \theta,k,p > 0$$

and this simplifies to the Gamma distribution if $p = 1$. We fit this distribution model to the empirical datasets using

[1]The average file size for the archival storage is also much bigger than the non-archival storage (29.2MB vs. 700KB). The archival storages also contain both more files and a larger volume of files, the number of files ranging from 5.3M to 13.7M, and utilization volume ranging from 69TB to 305TB for the archival storage, but only from 1.5M files to 11.3M files and from 1.2TB to 9.2TB for non-archival storage. Further statistics are documented in [2].



(a) CDF and CCDF from HPC sites [8] (b) Gamma and Gen. Gamma fitting (c) Spline fitting

Figure 1: Archival and Non-Archival File size distributions

generalized nonlinear least-squares (`gnls` in `R` [4]) to find scale (θ) and shape (k, p) parameters that best matches the empirical data. Note that since the Generalized Gamma distribution simplifies to Gamma distribution when $p = 1$, our model automatically includes a traditional Gamma distribution. However, to evaluate the difference between Generalized Gamma and Gamma distributions we also fixed $p = 1$ to find the best Gamma distribution fit (see Figure 1(b)). We find that the Generalized Gamma model was much better at fitting the data for large files than the Gamma model, as shown in the CCDF plot[2].

However, we also wanted to ensure our measurements were robust. We can model variation in the typical file size distribution by considering the covariance across the different data sets – `arsc-nanu1`, `arsc-seau2`, `arsc-seau1` and `pnnl-nwfs` – to produce appropriate outlier distributions (Confidence Intervals (CIs)) so as to model variation amongst the archives. This covariance provides both a mean and standard deviation for the model parameters, i.e., θ, k and p. Assuming a normal distribution for the parameters, as is assumed under nonlinear least squares, we can then bootstrap a number of parameter values $(\theta_i^B, k_i^B, p_i^B), i = 1, \ldots, N$ which give N bootstrapped CDFs, $F_i^B(x)$. Each bootstrapped CDF is defined for each of our file sizes, e.g., 2KB, 8KB, etc, so we create *sorted lists* of CDF values for each file size, i.e., sort $F_i^B(2\text{KB})$ over $i = 1, \ldots, N$. From each of the sorted lists we can identify the $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ percentiles to build $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ functions. Note that, using this procedure, it is possible that $F^{\frac{\alpha}{2}}(x_1) > F^{\frac{\alpha}{2}}(x_2)$ for $x_1 < x_2$, i.e., $F^{\frac{\alpha}{2}}(x)$, which is not a valid CDF, so we increase the "width" of our envelope slightly:

$$\tilde{F}^{\frac{\alpha}{2}}(x_i) = \min\{0, F^{\frac{\alpha}{2}}(x_i), F^{\frac{\alpha}{2}}(x_{i+1})\}$$
$$\tilde{F}^{1-\frac{\alpha}{2}}(x_i) = \max\{F^{1-\frac{\alpha}{2}}(x_{i-1}), F^{1-\frac{\alpha}{2}}(x_i)\}$$

for $x_i = 2^i\text{KB}, i = 1, 2, \ldots$ and $F^{1-\frac{\alpha}{2}}(x_0) = 0$.

[2]We no longer consider the non-archival data as that is not the focus of this research.

We use this bootstrapping procedure to produce 95% and 99% CI envelopes for the typical file size CDFs. This means we are 95% and 99% confident respectively that the actual typical file size distribution lies within the envelopes. The "lower" bounds are the fit under the median CDF fit (more large files), and the "upper" bounds are the fit above the median CDF fit (more small files).

With both the Gamma and Generalized Gamma distribution, both the 95% and 99% envelopes produced large probabilities of files with size >64M which was not observed in the empirical data. As stated before, the CDF plot in Figure 1(b) shows that the best Gamma distribution model did not fit the data that well and the CCDF shows that the fit to the tail was very poor. Conversely, the best Generalized Gamma distribution model did fit the data well, including the tail of the distribution. However, the envelopes produced when using the Gamma distribution match the (poorly fitting) median Gamma distribution. By contrast, the envelopes produced using the Generalized Gamma do not match the median Generalized Gamma distribution well at all and (similar to the Gamma models) produce distributions with the probability of a large file (>64M) much higher than the empirical archival distributions. These results show that neither the Gamma nor the Generalized Gamma distributions provide good envelopes for robust benchmarking.

We considered several other options to model the typical file size distribution including a composite distribution function with a Lognormal distribution to fit the CDF for smaller file sizes that then switches to a Pareto-tail function to fit the CCDF for large file sizes. Agrawal et al. [6] found this function to be effective for matching both small and large file sizes, but their datasets arose from Windows Desktop machines. The fit for the archival storage data was poor (in fact worse than for the Generalized Gamma distribution).

Our final alternative was to use a Spline-based CDF to fit the data. A spline is a set of piecewise polynomials that join smoothly at "knot" points to form the overall function. We used a cubic B-spline [9] and evenly spaced knots at

quantiles. We constrained the spline coefficients to ensure we built a monotonically non-decreasing function. We used generalized nonlinear least-squares to determine the best coefficients for each piece of the spline, because the empirical CDF within each individual archive is correlated. Specifically, a continous AR(1) process was assumed for the correlation, and this was found to produce patternless normalized residuals.

The CDF model produced by fitting the spline matched the data as well as the Generalized Gamma. Moreover, it produced 95% and 99% envelopes that represent the variability in the data and also match the median CDF, i.e., that provide realistic CDFs while still incorporating the variability observed in the empirical data. Figure 1(c) compares the spline model to the empirical data. Note that, in some cases, the empirical CDFs may fall outside the 95% or 99% envelopes. This is because we are 95% or 99% confident that the typical file size distribution is within the respective envelope, but there may be some outlier CDFs that exhibit unusual behavior and hence are not within the envelope. However, without any extra information, these envelopes should be used to provide a robust benchmark for typical archival storage performance. The envelope not only gives both the most likely CDF for typical file size, but also lower and upper limits that represent the possibility that the actual typical file size distribution may have more small files or more large files respectively.

2.3 Generating a typical workload

Once we determined the best envelope for the typical file size CDF, we have 3 file size CDFs to use for benchmarking. We used each of these distributions as input to `FFSB` [1], a file benchmark tool. In order to produce a consistent benchmark amongst all setups, we slightly modified the tool so that we can scale the distributions for different total storage capacities. We refer to a generated set of files of particular sizes and frequencies as a *fileset*.

Each CDF gives us $F(x) = \Pr(X \leq x)$, but we know that $\Pr(X = x) = 0$ for all $x \notin \{x_i \equiv 2^i\text{KB} | i = 1, 2, \ldots\}$. Thus, $\Pr(X = 2\text{KB}) = \Pr(X = x_1) = F(x_1)$, $\Pr(X = 4\text{KB}) = \Pr(X = x_2) = F(x_2) - \Pr(X = 2\text{KB})$, and so on for $\Pr(X = x_i), i \geq 2$. Using these probabilities we can produce the appropriate fileset of files with sizes 2^iKB for any total file count C or total file volume V.

Given C, $c_i = \Pr(X = x_i) \times C$ so $C = \sum_i c_i$ (round to nearest integer). The volume of files (in the fileset) of size 2^iKB is $v_i = 2^i \times \Pr(X = x_i) \times C$, so the total file volume is $V = \sum_i v_i$. Conversely, given V, we know that $V = \sum_i 2^i \times \Pr(X = x_i) \times C$ and dividing by C gives the average file size $\mathrm{E}[X] = \frac{V}{C} = \sum_i 2^i \times \Pr(X = x_i)$. We know V and can calculate $\mathrm{E}[X]$ from $\Pr(X = x_i)$. Thus, $C = \frac{V}{\mathrm{E}[X]}$ and $c_i = \Pr(X = x_i) \times C = \Pr(X = x_i)\frac{V}{\mathrm{E}[X]}$.

We can use this formula to generate the c_i's for testing a given percentage of a filesystem's capacity (i.e., its *capacity utilization*). For example, to produce a fileset that tests 10% of a 2TB filesystem (200GB) we set $V = 200 \times 2^{20}$KB and produce a fileset with c_i files of size 2^iKB for $i = 1, 2, \ldots$.

One of the important characteristics of a HDD is that the total volume of files either being stored or retrieved are often affected by mechanical characteristics of the disk. For instance, during a sequential write, data in a new disk will first be stored in the outermost edge (clusters) of the platter and, as more capacity is utilized, data is stored towards the inner edge of the platter. This means that the performance – both write and read – will decrease as the disks become full, and thus per-disk MB/s declines with capacity utilization.

We want to measure the archival tier components in an environment close to that they would experience in a large storage archive. A real-world archival storage tier contains many disks, so the workload experienced by an individual disk within the tier will be different than that experienced by the same disk in isolation. For example, if a 2TB disk is 10% utilized, then the total fileset volume is 200GB. Using that same fileset on a 10 disk system, without the %, will result in each disk experiencing only 20GB workload, so each individual disk's performance is better as only the outer edge of each disk will be used. However, if the 10 disk system is 10% utilized, then the system volume is 20TB, so the fileset volume is 2TB and each individual disk of the 10 will experience an approximate workload of 200GB – both sequential-write and random-read – and the performance comparison is fair and consistent. Consistency of workloads for benchmarking is widely ignored when considering an increased number of disks. Usually the fileset (and hence its total volume) is fixed and, as the number of disks increases, the performance improves simply because of the I/O performance improvement per disk due to the decreasing *per disk* workload. This inconsistent benchmarking ignores any performance changes due to the system itself and, as a consequence, disregards the effects of large capacity, caching, multiple disks and multiple disk arrays.

To address this inconsistency, we use capacity utilization instead of fixed volume filesets, i.e., what % of the total storage capacity is being utilized, and, in addition to our 3 file size CDFs, experiment with different levels of utilization in our benchmarking.

3. PERFORMANCE COMPARISONS

3.1 Experiments on an ext4 filesystem

Our hypothesis is that the benchmarks produced by filesets generated by our CDFs will show performance results that best match the performance of benchmarks from the empirical archival file size distributions. We compare our benchmarks to benchmarks generated by the Generalized Gamma distribution models, i.e., that it is important to have a good envelope, not just a good median fit. We also compare our benchmarks to traditional benchmarks that use a small mixture of file sizes and I/O requests, rather than an entire CDF for file sizes.

Our test system consist of Intel Xeon 5630 CPU with 18GB RAM, SuperMicro X8DTN+-F (Intel X58, 5520 Chipset) and LSI 2108 (512MB RAM) RAID controller. The storage device is configured with 12TB – 6×2TB WDC `WD20EAR` accessed via the RAID controller (RAID 0 in write-through mode with 8K stripe and direct IO).

We used the following distributions for generating filesets:

1. Archival empirical distributions:
2. Fitted distributions: median, 95% lower and 95% upper envelopes from the Generalized Gamma and Spline distributions;
3. Fixed file size distributions: as shown in Table 2 and Figure 2(b). These distributions represent traditional

Capacity Utilization	Empirical archival distributions					Fitted models					
						Generalized Gamma			Spline		
	arsc-nanu1 E[X] = 14.8MB	arsc-seau2 = 30.2MB	arsc-seau1 = 43.8MB	pnnl-nwfs = 27.9MB	avg. = 29.2MB	median = 24.5MB	lower = 1.7GB	upper = 3.8MB	median = 25.8MB	lower = 28.7MB	upper = 8.1MB
120GB (1%)	55.4	58.3	69.8	58.7	60.6	61.5	51.3	47.2	66.1	60.1	59.1
600GB (5%)	42.3	35.9	43.6	36.2	39.5	41.9	4.8	34.7	41.7	39.8	39.9
2.4TB (20%)	35.9	32.9	41.3	31.2	35.3	32.7	2.7	36.0	34.3	38.6	34.7
4.8TB (40%)	31.1	37.6	36.8	29.7	33.8	33.8	2.0	36.0	35.5	33.2	31.9

Table 1: Random-read MB/s of empirical archival distributions and fitted models

benchmarks where a limited mix of averaged file sizes are used[3].

We generate filesets to use 1%, 5%, 20% and 40% capacity utilization respectively. Each fileset is then tested on an ext4 filesystem using following steps: 1) sequential-write (and then sync) of the entire fileset to the storage; 2) random-reads (128, 256 and 512 threads) from the files in the storage until all files have been retrieved at least once and reading has been happening for at least 30 minutes; and 3) recreate the ext4 partition (reformat) for the next fileset. Each test run is completely isolated and we drop all caches between the steps.

For sequential-write, we observed no obvious differences among the filesets for a particular capacity utilization. We also observed no obvious difference when changing the number of threads for random-reads. All results are for 512 threads hereafter. For random-reads, there were clear differences between the distributions. Table 1 and Table 2 show the results for the CDF benchmarks and fixed file size benchmarks respectively. The difference between the average of the empirical benchmarks and the other benchmarks (both fitted and fixed file size) are shown in Figure 2.

The performance of all benchmarks drops as the capacity utilization increases because the disks are not able to maintain I/O at the edge of the platter, and also because the disks are randomly accessing much larger cluster areas. The match for the Generalized Gamma 95% lower envelope also deteriorates. This is because the Generalized Gamma 95% lower envelope does not fit the empirical data well for large file sizes, so the filesets produced will have too many large files deteriorating the performance. This behavior is not observed in the empirical data, nor in the Spline 95% upper envelope which models the empirical data more accurately.

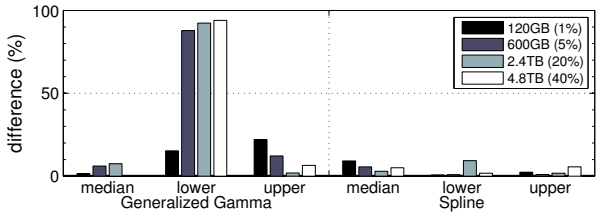
The match for all of the fixed file size benchmarks is generally poor, but both the 32MB and 64MB provide reasonable matches. This is because they are the closest to the average empirical file size of 29.2MB, but their accuracy is not as good as the Spline envelope and, because their file size is fixed, they do not represent the variation observed in typical archival workloads.

Note that we have also examined the performance difference between the empirical non-archival and archival distributions. We found that non-archival distributions also have large performance differences. For instance, at 5% capacity utilization, non-archival distributions performed on average 27.3MB/s whereas archival distributions performed on average 39.5MB/s; 31% performance difference.

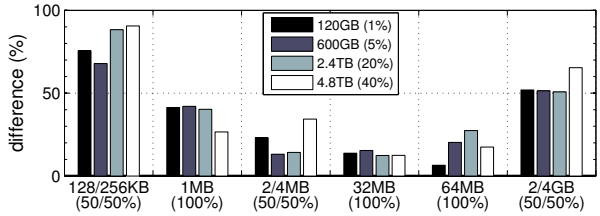
[3]The file size mixtures selected are chosen from observed average file sizes, e.g., 128/256KB, 1MB – from increasing the average file sizes observed 108KB (2000) to 318KB (2009) [7,11] and from the non-archival data; 2/4MB – from average file sizes observed over various years [8]; 32MB – from the archival data; and 2/4GB – from large binaries and images.

Cap. Util.	Fixed file size model					
	128/256KB (50/50%)	1MB (100%)	2/4MB (50/50%)	32MB (100%)	64MB (100%)	2/4GB (50/50%)
1%	14.8	35.5	46.6	52.2	56.6	92.0
5%	12.7	22.9	34.3	45.6	47.5	19.2
20%	4.1	21.1	30.3	39.7	45.0	17.4
40%	3.2	24.8	22.2	38.0	39.7	11.7

Table 2: Random-read MB/s of fixed file size models



(a) Empirical archival distribution vs. fitted model



(b) Empirical archival distribution vs. fixed file size model

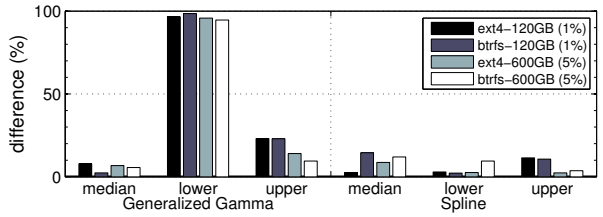
Figure 2: Random-read MB/s differences in %

3.2 Experiments on a Ceph filesystem

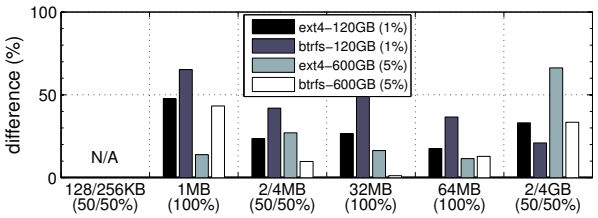
Overall setups are similar to the previous test and all the Ceph (v0.34) processes (MDS, MON and OSD) are run on the single machine. Two filesystems are measured – a single OSD running on top of ext4, and on top of btrfs. All the Ceph configurations are set to a default minimal setup, with 2GB journal file size and no data replication for both MDS and OSD. We then used an additional machine – the same hardware specification – as the Ceph Client mounting the Ceph filesystem to run the benchmark. Two machines are connected using a *bonded* 4×Gb/s Eth NICs and the throughput between the two are measured as 3.4Gb/s.

Generally both ext4 and btrfs have the similar results to the previous benchmark in Section 3. A few differences we noticed were that the btrfs performed on average 51% faster random-read than the ext4 (70MB/s btrfs vs. 34MB/s ext4). Also, possibly due to Ceph's object sync mechanism for the distributed network environment, the sequential-writes were slow (70–80MB/s Ceph vs. 450–500Mb/s local ext4). Because of this, we were not able to complete 20% and 40% capacity utilization due to time constraints.

Overall results were similar to Section 3; the Spline's fits have smaller MB/s difference than the Generalized Gamma's fit (Figure 3(a)) and the fixed file size fit (Figure 3(b)). Sim-



(a) empirical archival distribution vs. fitted model



(b) empirical archival distribution vs. fixed file size model

Figure 3: Ceph Random-read MB/s differences

ilarly, the observed large differences of the fixed file size show that both ext4 and Ceph filesystems have different behaviors for the 1% and 5% filesets – with no obvious trend amongst them (e.g., 2/4MB, 32MB and 64MB). Note that there are no results for fixed file size of 128/256KB in Figure 3(b) because Ceph was not be able to complete the sequential-write; this is most likely due to Ceph's default setup, specifically the MDS's object cache limitation.

Both plots confirm that the Spline envelopes "match" the empirical behavior. Contrastingly, we observed particularly unrealistic performance from for Generalized Gamma's lower envelopes. The empirical distributions may change over time or new data may be come available, requiring our process to be repeated. Without such extra information, the envelopes we produced from the empirical archival distribution – lower (more large files) and upper (more small files) – provide a robust envelope for those distributions. Our robustness benchmarks show that using Spline envelopes results in the most realistic results that incorporate variability.

4. CONCLUSION

Our experiments show that traditional benchmarking is not appropriate for archival storage. The range of possible file sizes and variation in performance can only be accurately modeled using a CDF of file sizes, not using a small subset of fixed file sizes. However, our experiments also show that not only is an accurate representation of the typical archival workload necessary, but for robust benchmarking, i.e., benchmarks that take into account possible variation in that workload, the envelope must also accurately represent the variation observed. If variation is introduced due to the modeling of the workload, then resultant measurements will be also be inaccurate (and possibly worse than the fixed file size measurements).

Our observations require that benchmarks are run for varying capacity utilizations and for a reasonable duration. If benchmarks are run for a short duration then larger files may never be read, but, given that files exists in an archive for a long time, it is likely that larger files will be read at some stage. Benchmarks that do not accurately model the presence of large files will not provide good predictions of actual archive performance.

Our benchmarking process could be improved by using empirical measurements of temporal access patterns combined with the file size distribution. These access patterns could be analyzed and modeled in an analogous way using traditional distributions or our Spline-based approach as appropriate. Robust benchmarks could be ensured by incorporating the variation in these observed access patterns. However, empirical measurements of temporal access patterns for typical archival storage tiers do not currently exist in literature, so our use of the file size distribution was the best viable approach for our robust benchmarks.

With an increasing number of options for archival storage, architects need to carefully consider the kind of workloads that their storage system will experience. Our benchmarks provide a robust measurement tool for typical archival workloads. Using these benchmarks will help storage architects with their design decisions and reduce occurrences of overprovisioning due to uncertainty of users' workloads.

5. REFERENCES

[1] Flexible File System Benchmark. `http://sourceforge.net/projects/ffsb/`.

[2] fsstat - Static Survey of File System Statistics. `http://www.pdsi-scidac.org/fsstats/`.

[3] SNIA - I/O Trace Data Files. `http://iotta.snia.org/traces/`.

[4] The R Project for Statistical Computing. `http://http://www.r-project.org`.

[5] I. Adams, E. L. Miller, and M. W. Storer. Analysis of workload behavior in scientific and historical long-term data repositories. Technical Report UCSC-SSRC-11-01, University of California, Santa Cruz, 2011.

[6] N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. In *FAST*, pages 125–138, 2009.

[7] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. In *FAST*, pages 31–45, 2007.

[8] S. Dayal. Characterizing hec storage systems at rest. Technical Report CMU-PDL-08-109, Carnegie Mellon University Parallel Data Lab, 2008.

[9] T. J. Hastie and R. J. Tibshirani. *Generalized additive models*. London: Chapman & Hall, 1990.

[10] A. Iamnitchi, S. Doraimani, and G. Garzoglio. Workload characterization in a high-energy data grid and impact on resource management. *Cluster Computing*, 12(2):153–173, 2009.

[11] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *FAST*, pages 1–13, 2011.

[12] E. L. Miller and R. H. Katz. An analysis of file migration in a unix supercomputing environment. In *USENIX Winter*, pages 421–434, 1993.

[13] N. Park, W. Xiao, K. Choi, and D. J. Lilja. A statistical evaluation of the impact of parameter selection on storage system benchmarks. In *SNAPI*, 2011.

[14] E. W. Stacy and G. A. Mihram. Parameter estimation for a generalized gamma distribution. *Technometrics*, 7(3):349–358, 1965.

[15] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. A nine year study of file system and storage benchmarking. *TOS*, 4(2), 2008.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *OSDI*, pages 307–320, 2006.

[17] A. Wildani, E. L. Miller, and L. Ward. Efficiently identifying working sets in block i/o streams. In *SYSTOR*, page 5, 2011.