2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing

# Metadata Traces and Workload Models for Evaluating Big Storage Systems

Cristina L. Abad*†‡, Huong Luu*, Nathan Roberts†, Kihwal Lee†, Yi Lu* and Roy H. Campbell*

*University of Illinois at Urbana-Champaign; †Yahoo! Inc.
*{cabad,luu1,yilu4,rhc}@illinois.edu; †{nroberts,kihwal}@yahoo-inc.com

***Abstract*—Efficient namespace metadata management is increasingly important as next-generation file systems are designed for peta and exascales. New schemes have been proposed; however, their evaluation has been insufficient due to a lack of appropriate namespace metadata traces. Specifically, no Big Data storage system metadata trace is publicly available and existing ones are a poor replacement. We studied publicly available traces and one Big Data trace from Yahoo! and note some of the differences and their implications to metadata management studies. We discuss the insufficiency of existing evaluation approaches and present a first step towards a statistical metadata workload model that can capture the relevant characteristics of a workload and is suitable for synthetic workload generation. We describe Mimesis, a synthetic workload generator, and evaluate its usefulness through a case study in a *least recently used* metadata cache for the Hadoop Distributed File System. Simulation results show that the traces generated by Mimesis mimic the original workload and can be used in place of the real trace providing accurate results.**

***Index Terms*—metadata; HDFS; storage; MDS; Big Data**

## I. Introduction

Large-scale file systems in the Internet services and high-performance computing communities already handle Big Data: Facebook has 21PB in 200M objects and Jaguar ORNL has 5PB [2]. Stored data is growing so fast that exascale storage systems are expected by 2018-2020, by which point there should be over five hundred 10PB deployments [11].

Large-scale storage systems frequently implement a separation between data and metadata to maintain high performance [6]. With this approach, the management of the metadata is handled by one or more namespace or metadata servers (MDSs), which implement the file system semantics; clients interact with the MDSs to obtain access capabilities and location information for the data [21]. Larger I/O bandwidth is achieved by adding storage nodes to the cluster, but improvements in metadata performance cannot be achieved by simply deploying more MDSs, as the defining performance characteristic for serving metadata is not bandwidth but rather latency and number of concurrent operations that the MDSs can handle [6, 10]. For this reason, novel and improved distributed metadata management mechanisms are being proposed [14, 15, 21]. However, realistic validation and evaluation of these designs is not possible as no adequate traces or workload models are available (see § III).

‡ Also affiliated with Facultad de Ingeniería en Electricidad y Computación (FIEC), Escuela Superior Politécnica del Litoral (ESPOL), Campus Gustavo Galindo, Km 30.5 Vía Perimetral, Guayaquil–Ecuador.

We argue for the need of Big Data traces and workload models to enable advances in the state-of-the-art in metadata management. Specifically, we consider the case of namespace metadata traces. We define a **namespace metadata trace** as a storage system trace that contains both a snapshot of the namespace (file and directory hierarchy) as well as a set of events that operate atop that namespace (e.g., open a file, list directory contents, create a file). I/O operations need not be listed, making the trace smaller. Namespace metadata traces can be used to evaluate namespace management systems, including their load balancing, partitioning, and caching components. Our methodology is presented in § II.

We focus on namespace metadata management because it is an important problem for next-generation file systems. However, other types of research that need information about realistic namespaces and/or realistic workloads or data access patterns in Big Data systems could benefit significantly from access to these traces and models: job scheduling mechanisms that aim to increase data locality [22], dynamic replication [1], search in large namespaces [12], schemes that want to treat popular data differently than unpopular data [7], among others.

Publicly available storage traces do not meet our definition of a *namespace metadata trace* since they do not contain a snapshot of the namespace. Due to the heavy-tailed access patterns observed in Big Data workloads, this means that the trace-induced namespace will not contain a large portion of the namespace (i.e., that portion that was not accessed during the storage trace). More importantly, existing traces are not representative of Big Data workloads and are frequently scaled up through ad-hoc poorly documented mechanisms (see § III).

In § IV, we present a new (work-in-progress) statistical model for namespace metadata workloads and describe Mimesis, a synthetic workload generator based on our model. Preliminary results (§ V) suggest that our model can be used to generate synthetic traces that mimic the original workload: the results obtained using Mimesis have a low *root mean squared error* (within 5.82%), outperforming other prior approaches.

In § VI, we discuss the related work; in § VII we conclude.

## II. Datasets and methodology

We support our arguments through an analysis of: a Big Data *namespace metadata trace* from Yahoo and two traces used in prior work (Home02 and EECS in Tables I-II; which are the most recent public traces used in the papers we surveyed [2]). The Big Data trace comes from the largest

978-0-7695-4862-3/12 $26.00 © 2012 IEEE
DOI 10.1109/UCC.2012.27

TABLE I
TRACES USED BY RECENT PAPERS THAT PROPOSE NOVEL METADATA MANAGEMENT SCHEMES; REFERENCES AVAILABLE IN [2].

| Trace name | Year | Source description |
|---|---|---|
| Sprite | 1991 | One month; multiple servers at UC Berkeley. |
| Coda | 1991-3 | CMU Coda project, 33 hosts running Mach. |
| AUSPEX | 1993 | NFS activity of 236 clients, during one week in UC Berkeley. |
| HP, INS (HP), RES (HP) | 2000 | Hp-UX traces, 10+ days, 13+ clients, 500GB, 94.7 million requests, 0.969 million files. |
| Home02 (Harvard) | 2001 | From campus general-purpose servers; 48GB. |
| EECS (Harvard) | 2001 | NFS trace; `home` directory of computer science department; 9.5GB. |

TABLE II
TRACES ANALYZED IN THIS PAPER; AOA: AGE AT TIME OF ACCESS. # FILES INCLUDES FILES CREATED OR DELETED DURING THE TRACE.

| Trace | # Files | Used storage | Mean interarrival time (milliseconds) | AOA (median, in seconds) |
|---|---|---|---|---|
| Yahoo | 150M | 3.9 PB | 1.04 | 267 |
| Home02 | > 1M | 48 GB | 243.80 | 4682 |
| EECS | > 1M | 9.5 GB | 27.20 | 1228 |

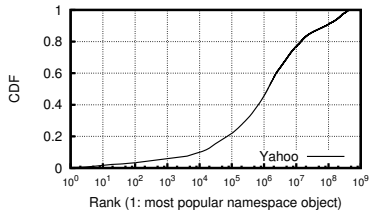



Fig. 1. Cumulative distribution function of the popularity of namespace objects (files/directories) in Yahoo trace, for those accessed (metadata event) at least once during trace; 3.28% of objects account for 80% of accesses.

Hadoop cluster at Yahoo (4000+ nodes, HDFS); this is a production cluster running data-intensive jobs like processing advertisement targeting information. The *namespace metadata trace* has a snapshot of the namespace (04/30/2011) obtained with Hadoop's Offline Image Viewer tool, and a 1-month trace of namespace events (05/2011) obtained by parsing the namenode (MDS) audit logs. See [3] for workload details.

Storage system workloads are multi-dimensional [8, 19] and can be defined by several characteristics like namespace size and shape, arrival patterns, and temporal locality patterns. In this paper, we discuss of some of these dimensions where appropriate. One of these dimensions is the temporal locality present in the workload, which we measure through the distribution of the age of a file at the time it is accessed (*AOA*). For every operation (namespace metadata event), we calculate how old the file is and use this information to build a cumulative distribution function (CDF) that represents the workload in this dimension. We chose this dimension because it is one that is very relevant to namespace metadata management, since the temporal locality of the workload has an incidence in mechanisms like load balancing, dynamic namespace partitioning/distribution, and caching.

In this paper, an *access* to an object refers to an access through a *namespace metadata event*. For example, getting the attributes of a file constitutes a metadata access.

## III. LIMITATIONS OF EXISTING APPROACHES

We surveyed the papers published in the last five years that propose novel namespace metadata management schemes and identified their evaluation methodology. The approaches fall into one of three categories listed in Table III. We focus on approach (iii), but briefly discuss (i) and (ii) first.

*(i) Metadata-intensive microbenchmarks:* While many I/O benchmarks exist, they do not evaluate the metadata dimension in isolation [18], making them inadequate for metadata management studies. Mdtest and metarates are metadata-intensive microbenchmarks; however, they do not attempt to model real workloads. For example, one can use mdtest to issue a high-rate of creates (of zero-sized files) in a random namespace, but not to do creates modeled after real workloads and atop a realistic namespace.

*(ii) Application benchmarks:* Synthetic benchmarks exist and real non-interactive applications can be used too. However, the full workload of a system is typically a combination of different applications and client usage patterns that, as a whole, can differ from the individual application workloads.

### A. Limitations of existing traces

We analyzed the traces used in prior papers (Table I), as well as the storage traces available for download at the Storage Networking Industry Association (SNIA) trace repository, and identified these limitations in the context of namespace metadata management studies for next-generation storage systems:

*1) No public petascale traces are available:* Sub-petascale traces are often scaled up in some way; the modification of the traces, if done through ad-hoc poorly documented mechanisms, makes the results difficult to reproduce and raises questions about the validity of the results. Working at a smaller scale is not always adequate since inefficiencies in design/implementation may only be evident at scale.

*2) No traces include both a namespace snapshot and a trace of operations on that namespace:* If no namespace snapshot is included with trace, the researcher must rely on the trace-induced namespace[1] (i.e., that portion of the namespace that is accessed during the trace). The trace-induced namespace can be significantly smaller—due to the heavy-tailed access patterns in which some files are rarely, if ever, accessed (see Figure 1)—and may have a different form than the full namespace (see Figures 2 and 3). In other words, the trace-induced namespace is a bad predictor of the actual namespace, and the omission of a significant portion of the files and directories in an evaluation could affect its results.

Additionally, any study that requires knowledge only available in the snapshot (e.g., file age or size) would be limited or biased if no snapshot is available. For an example, consider the significant difference in AOA when calculated with full

[1]Or, create a namespace from a model [4], if available.

TABLE III
EVALUATION APPROACHES USED IN PRIOR NAMESPACE METADATA MANAGEMENT PAPERS; LIST OF REFERENCES AVAILABLE IN [2].

| Evaluation mechanism | Description |
|---|---|
| *(i) Metadata-intensive microbenchmarks* | *Test a specific part of the system and do not attempt to represent a realistic workload.* |
| mdtest | Multiple processes create/stat/delete files/directories in shared or separate directories. |
| metarates | MPI program that coordinates file system accesses from multiple clients. |
| self-designed | Non-standard scripts/programs that perform some sequence of namespace operations. |
| *(ii) Application benchmarks* | *Provide real or synthetic application-based workloads.* |
| Checkpoint, SSCA, IMAP build, mpiBLAST | Multi-process real or pseudo-real applications, typically metadata intensive. |
| *(iii) Trace-based approaches* | *Aim at looking at the full workload of the system.* |
| Simulation | Simulator designed by authors; traces may be scaled up. |
| Replay | Replay of real traces, scaled up to simulate a larger system. |

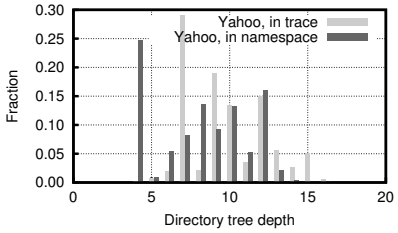



Fig. 2. Fraction of files at each depth in hierarchy; a high percentage of files are stored at depth 4 but rarely accessed.

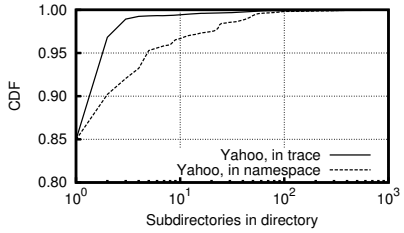



Fig. 3. In trace-induced namespace, non-leaf subdirectories appear to have fewer children than in snapshot.

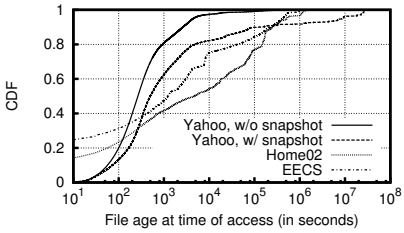



Fig. 4. File age at time of access for a Big Data and two enterprise storage traces; there's significant difference in temporal locality between workloads.

knowledge of the ages of the files and with partial knowledge of the ages (without a snapshot, we can only know the age of the files created during the trace of events) shown in Figure 4.

*3) No Big Data traces are available*[2]*:* Big Data workloads may differ considerably from more traditional workloads. For example, consider the age of a file at the time it is accessed, which is a measure of the temporal locality on a trace based on previous Big Data observations. Figure 4 shows the difference in temporal locality between two traditional traces and a Big Data trace[3]. In the Yahoo trace, most of the accesses to a file occur within a small window of time after the file is created; this is typical of MapReduce pipelines. In contrast, in Home02 and EECS files remain popular longer.

### B. Lack of metadata workload models

When adequate traces are not available, workload models can be useful by enabling researchers to generate synthetic traces or modify existing traces in a meaningful way. While several models have been proposed to describe certain storage system properties (e.g., directory structure in namespace snapshots [4]), to the best of our knowledge no work has been published that proposes a model that combines a namespace structure and a (metadata) workload on that trace. This lack of adequate models make it hard for researchers to design their own synthetic workloads or to modify existing traces to better fit access patterns of large scale storage systems.

[2]Application workloads exist, such as Hadoop's Gridmix for MapReduce and the simulation traces from Sandia National Labs. While these traces are useful, they do not represent the full load observed by the *storage* system.

[3]For traces without a namespace snapshot (Home02 and EECS), it is not possible to know the age of files that were created before the trace of events. To enable comparison between traces, for the Yahoo trace we plotted the AOA with full information of the file ages (with snapshot) and after ignoring those files that were not created during the trace (without snapshot).

## IV. TOWARDS A METADATA WORKLOAD MODEL

We present a model for namespace metadata traces that *captures the relevant statistical characteristics of the workload and is suitable for synthetic workload generation*.

We begin the description of our model by formalizing our definition of a namespace metadata trace as follows.

*Definition 1:* In a distributed file system that separates metadata management from data storage, a **namespace event** is a client request received by the namespace metadata management subsystem, which corresponds to a namespace request. Typical requests include those to create, open, or delete a file, creating or deleting a directory, and listing the contents of a directory. The format of each event record in a trace may vary between systems, but its simplest form is: <timestamp, operation, operation parameters>.

*Definition 2:* Let $S_t$ be a snapshot of the namespace at time $t$ and $E = \{e_i ... e_k\}$ be the list of all the namespace events as observed by the namespace server(s). Then, a **namespace metadata trace**, $T_{t1-t2}$, is a trace that contains a snapshot $S_{t1}$ of the namespace at a time $t_1$, and a set $E_{t1-t2}$ of events $e_{t1} ... e_{t2}$, where $t2 = t1 + \delta$, $\delta \gg 0$ and $e_j \in E_{t1-t2}$ iff $e_j \in E$, $j \geq t1$ and $j \leq t2$.

Given $T_{t1-t2}$, we can model its workload as a set of parameters that together define the namespace (modeled after $S_{t_1}$) and the workload of metadata events (modeled after $E_{t1-t2}$). Our current implementation contains the statistical parameters listed in Table IV, which is the set of probability distributions (empirical or fit to a known distribution) that describe the most relevant[4] statistical properties of the namespace, namespace as used in the workload, and of the workload itself.

[4]Set of parameters chosen after a literature review in storage namespace and workload modeling; other parameters can be added in the future.

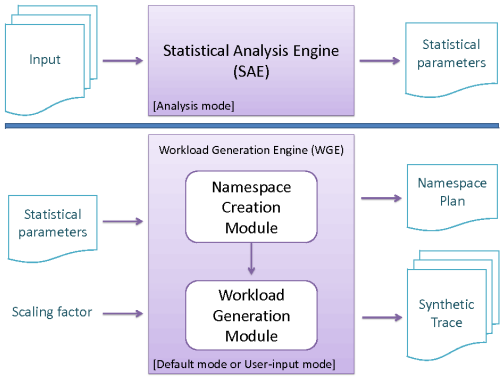



Fig. 5. Block diagram of our synthetic workload generator, Mimesis.

We have an implementation, called Mimesis (see Figure 5), that contains a format in which the model can be stored, processed and shared, and two subcomponents: (1) the Statistical Analysis Engine (SAE), that takes $T_{t1-t2}$ and generates a configuration file with the model parameters, and (2) the Workload Generation Engine (WGE), that takes the configuration file and generates a synthetic metadata trace based on the model.

### A. Parameters

Two sets of parameters characterize the workload, regarding: namespace structure and workload characteristics, including access patterns and trace-induced namespace (Table IV).

The namespace structure is extracted from $S_{t1}$ and describes the shape and size of the namespace hierarchy tree. The number of directories and files describe the size of the namespace. The shape of the hierarchy tree is described by the following distributions: files at each depth, directories at each depth, files per directory and subdirectories per directory. The file size distribution is also extracted, which can be relevant to problems involving data block replication and placement.

The access patterns describe the relationship between the operations and the age of the files, which indirectly describe temporal locality. This is important, for instance, to namespace partitioning or metadata caching. The distribution of the file age at time of access (AOA) and age at time of deletion (AOD) are extracted and reproduced in synthetic workloads.

The workload-induced namespace describes the hierarchy of objects accessed in the trace. The shape of the hierarchy tree induced by the workload can be different from the snapshot when there is a long tail of rarely accessed files. The hierarchy of the accessed objects is relevant to metadata caching, for instance. In addition, the fraction of operation types and interarrival rate distribution are also extracted.

Two main contributors to *locality of reference* in file request streams are the *popularity distributions* and *temporal locality* present in the requestsIn our current implementation, we capture the temporal locality of the references, and keep track of file accesses to favor frequently accessed ones (except for *deletes*), as described later in this section.

TABLE IV
STATISTICAL PARAMETERS CAPTURED BY OUR MODEL.

| **Namespace characterization** |
|---|
| Number of directories *and* number of files |
| Distribution of files at each depth in namespace hierarchy |
| Distribution of directories at each depth in namespace hierarchy |
| Distribution of number of files per directory |
| Distribution of number of subdirectories per directory |
| Distribution of file sizes (in MB) |
| Distribution of ages of the files at $t_1$ |
| **Workload characterization** |
| Percentages of operation type in trace |
| Interarrival rate distribution |
| Distribution of operations observed at each depth in namespace |
| Distribution of files per depth in namespace, as observed in trace |
| Distribution of dirs. per depth in namespace, as observed in trace |
| Distribution of number of files per directory, as observed in trace |
| Distribution of number of subdirs. per dir., as observed in trace |
| Distribution of file age at time of access |
| Distribution of file age at deletion (i.e., file life span) |

TABLE V
MIMESIS INPUT AND OUTPUT FORMATS.

| **(a) HDFS namenode log record example** |
|---|
| 2011-5-18 00:00:00,134 INFO FSNamesystem.audit: ugi=USERID ip=<IP> cmd=listStatus src=/path/to/file dst=null perm=null |
| **(b) Input/output format: namespace** |
| File creation time stamp, full path, file size (-1 for directories) |
| **(c) Input format: metadata operations** |
| Time stamp, metadata operation, source, destination (for renames) |
| **(d) SAE output format for empirical distributions** |
| Item (discrete) or bin (continuous), count, fraction, CDF |
| **(e) WGE output format; stamp is relative to beginning of trace.** |
| Time stamp, metadata operation, source, destination (for renames) |

*Input Format and Parameter Extraction:* The SAE takes a format shown in Table V. We first convert a namespace metadata trace to this format before the SAE can process it.

Each of the distributions in Table IV takes value as either a known distribution or an empirical CDF. When extracting parameters, the SAE attempts to fit a known distribution to the measured values using R (MASS package). If the values pass the goodness-of-fit test (Kolmogorov-Smirnov), the known distribution becomes the value of the parameter; otherwise, an empirical distribution is built using the CDF obtained from the input data. If multiple distributions pass the goodness-of-fit test, the one with the smallest test statistic ($D$) is chosen.

### B. Generating synthetic traces

The WGE has two modules: namespace creation and workload generation. The former is used to generate a namespace hierarchy that maintains the same structure as the original. The latter is used to generate a synthetic trace that maintains the desired workload characteristics, operates on the namespace generated by the namespace module, and preserves the data access patterns extracted from the original trace.

*1) Namespace creation module:* Creates the namespace in two phases: the directory hierarchy is created in the first phase, the files in the second. The naive approach, creating files and directories in parallel, leads to a bias towards locating more files in the lower depths of the hierarchy (which would naturally be created first) [4]. Output format detailed in Table V(b).

*a) Creating the directory hierarchy:* The simplest approach is to iteratively create the directories starting from $depth = 1$, where $numTargetDirs \times percDirsAtDepth1$ directories are created; to decide how many directories to create in $depth + 1$ we can sample from the distribution of subdirectories per directory, once per every directory at the current depth; the iterative process continues until the current depth has zero directories. Unfortunately, the output of this approach does not accurately model the input namespace. Due to the high percentage of directories with 0 or 1 subdirectories ($68\%$ and $27\%$)[5], this iterative process creates a small shallow hierarchy in which the distribution of directories at each depth is not maintained (except for depth 1). Alternatively, we can: (a) use two independent distributions (directories at each depth and subdirectories per directory) and try to match both constraints at the same time, or (b) model them as a joint distribution in which both random variables are defined on the same probability space. We chose approach (a) because it generates a smaller, simpler model, favored by the principle of parsimony; the accuracy of this approach is studied in § IV-C.

With approach (a), we need to satisfy two constraints simultaneously: directories at each depth and subdirectories per directory. We model this as a *bin packing* problem:

- There's one bin for each depth of namespace hierarchy.
- The capacity of each bin is defined by multiplying the target number of subdirectories (by default, the same number of directories as in the input namespace) by the fraction of directories at each depth.
- The items being packed is a list of subdirectory counts obtained by sampling from the distribution of subdirectories per directory until the sum of the samples is greater or equal than the target number of subdirectories. If we exceed the target, the last sample is adjusted so that the sum of the samples does not exceed it.

Bin packing is a classic NP-complete problem. We use a greedy approach in which we first sort (in descending order) the list of subdirectory counts per directory that we want to pack. Each of these counts is packed as a single item with some specific weight (number of subdirectories); we refer to these as a *set of sibling directories*. We then pack each set of sibling directories in the worst bin (i.e., the one with the most free space). Obtaining a solution using a greedy approach would not typically work out; however, given the high percentage of directories with 0 or 1 subdirectories/children, which we pack last, finding a solution with this approach is feasible.

Once the bins have been packed, we iteratively assign subdirectory counts at each depth starting from depth 1 (the root of the hierarchy is created at the beginning of the process). The process then proceeds as follows: at depth $d$ we assign a parent to all the sets of sibling directories packed in the corresponding bin. For each set, a childless parent from $depth - 1$ is chosen. If no childless parent exists, a random parent at $depth - 1$ is chosen. Using this method, the distribution of directories per subdirectory may differ slightly from the target (see § IV-C).

*b) Creating the files:* The workload generation module chooses files to access according to their age. For this mechanism to work, every file should have a creation time stamp. When creating the namespace, each file is assigned an age randomly sampled from the distribution of file ages; the age is converted to a creation stamp at the end of the file creation process ($time\ stamp = max(sampled\ ages) - age$).

Creating the files has the same multiple constraint issue as directories: distributions of files at each depth and number of files per directory. We model this problem as a bin packing problem in the same way as before.

Finally, the file size is assigned by sampling from the distribution of file sizes.

*2) Workload generation module:* Generates the synthetic trace from the output of the namespace module and configuration parameters. It currently preserves the interarrival rates, distribution of operations at each hierarchy depth, percentages of operation types, and temporal locality (AOA and AOD); and the namespace is stressed as in the original trace.

For simplicity, in this section we refer to removal operations as *deletes*, creation operations as *creates* and other operations as *regular accesses*; this classification is applied regardless of whether the operation is performed on a file or directory.

We simulate the operations arriving at the namespace server as events in a discrete event simulation. Interarrivals are sampled from the interarrival distribution defined in the configuration parameters. Upon arrival of an event we make a weighted random selection of the operation based on the table of percentages of operation types.

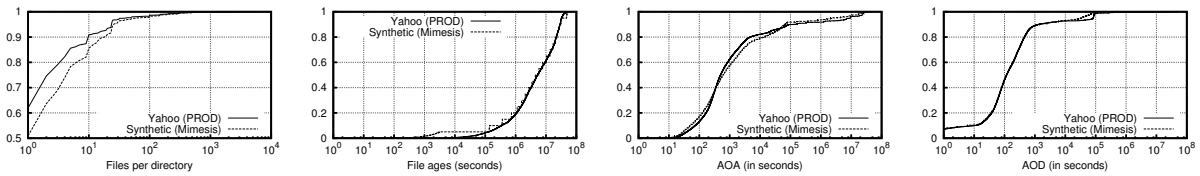
Next, a target depth is chosen by sampling from the distribution of operations at each depth of the hierarchy. Once the depth in the namespace hierarchy has been determined, the specific file or directory is chosen at that depth for *regular accesses* and *deletes*, or at the target $depth - 1$ for *creates*.

To preserve temporal locality of *regular accesses*, we use the age distribution of a file at the time of access (AOA) obtained by the SAE. We sample from the AOA distribution and obtain a target age. We search for the objects at the desired depth and choose the one with age closest to the target age. If more than one file approximates the desired age within some configurable *delta* (2000 milliseconds by default), we consider this to be a tie. Mimesis uses popularity information to break the ties as follows: the total number of accesses for each file is recorded during the workload generation. When a tie occurs, the file with the highest number of accesses (i.e., the most popular file) is chosen. The reasoning is that a popular file is more likely to keep receiving accesses than an unpopular file.

Similarly, we preserve the file life span or $AOD$ by sampling from this distribution in a *delete*. For the case of deletes, we break ties in the opposite manner: when a tie occurs, we choose the file with the least number of accesses (i.e., the least popular file). The reasoning is that a file that is unpopular is more likely to be deleted than a popular file.

Table V (e) shows the output format of the trace, which can be used for simulations and for testing real implementations.

[5]A 5-year study of file system metadata also found a high percentage ($> 80\%$) of directories with 0-1 subdirectories [5].




(a) Files per directory (b) File ages (c) Age of file at time of access (d) Age of file at time of deletion

Fig. 6. Accuracy of Mimesis for some of the statistical parameters (cumulative distribution functions) captured by our model; full results in Table VI.

TABLE VI
ACCURACY OF MIMESIS; TWO DIFFERENT WORKLOADS, AVERAGES ACROSS 10 TRIALS. THE KOLMOGOROV-SMIRNOV TEST STATISTIC ($D$) CONVERGES TO 0 IF SAMPLE COMES FROM TARGET DISTRIBUTION.

| Parameter | Yahoo, PROD | Yahoo, R&D |
|---|---|---|
| *Namespace characterization* | | |
| Files at each depth | 0.0001 | 0.0001 |
| Directories at each depth | 0.0002 | 0.0001 |
| Files per directory | 0.1105 | 0.1001 |
| Subdirectories per directory | 0.0158 | 0.0219 |
| File ages | 0.0478 | 0.0457 |
| File sizes | 0.0403 | 0.0419 |
| *Workload characterization* | | |
| Interarrival times | 0.0009 | 0.0008 |
| Operations at each depth | 0.0001 | 0.0001 |
| Files per depth, trace-induced | 0.0001 | 0.0001 |
| Dirs. per depth, trace-induced | 0.0001 | 0.0001 |
| Files per dir., trace-induced | 0.0998 | 0.0999 |
| Subdirs. per dir., trace-induced | 0.0106 | 0.0113 |
| Age at time of access | 0.0592 | 0.0617 |
| Age at time of deletion | 0.0444 | 0.0457 |

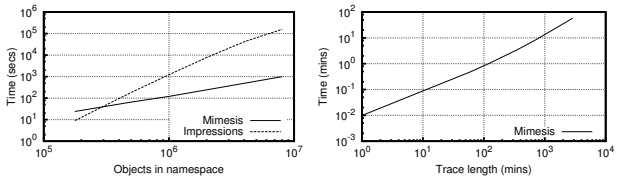
### C. Evaluation

We evaluate our approach's: (i) accuracy, (ii) performance, and (iii) usefulness (see § V).

To measure the **accuracy** of the synthetic traces generated by Mimesis, we use the Kolmogorov-Smirnov test statistic ($D$) (or, the maximum difference between the two CDF curves). Table VI shows the accuracy of the synthetic traces generated using our model parametrized after two traces: the Yahoo trace that has been described throughout this paper (PROD), and an additional 1-month trace (05/2011) from a 1900+ research and development cluster (R&D) at Yahoo. R&D is used for MapReduce batch jobs and ad-hoc data analytics/business intelligence queries (for details, see [3]). The synthetic traces Mimesis generates maintain the statistical properties of the original trace that are captured by our model with high accuracy (small $D$ values).

Figure 6 shows the real and synthetic trace CDFs, for four distributions with high $D$ values (PROD). For the files per directory CDF (highest $D$ value), the error comes from the cases for which no childless parent at $depth-1$ is found and a random parent is chosen instead.

To evaluate **performance**, we generated increasingly larger traces on a two quad-core PC (Xeon E5430, 2.66 GHz) with 16 GB RAM and a 1 TB SATA 7200 RPM disk. Mimesis is currently a single threaded Java program. Figure 7a shows that the namespace creation module outperforms Impressions [4] for large namespaces. Figure 7b shows how long the workload generation module takes to generate increasingly larger traces.




(a) Namespace creation (b) Trace generation

Fig. 7. Performance of the *namespace creation* and *workload generation* modules; number of objects = number of directories + number of files.

## V. APPLICATION: METADATA CACHE FOR HDFS

We evaluate Mimesis' **usefulness** with a case study: the need for a cache for the HDFS namespace server (MDS).

### A. Background

The HDFS has reached its scalability limits in large data-intensive systems [16]. One of the areas that can be improved is the MDS metadata handling. HDFS's design was inspired by the GFS [13], and inherited its design choice of keeping all metadata in memory. However, recent studies have shown that the memory footprint of an MDS server grows faster than the physical data storage [16], due to the file-count growth problem [13] which has emerged from an (incorrect) assumption that designing a file system with a large block size would encourage applications and users to generate a small number of large files. Furthermore, this design is wasteful considering that the access patterns in HDFS can show a long tail of infrequently accessed files (see Figure 1).

A common approach to this problem is to cache the popular metadata in memory and keep the rest in secondary storage. We evaluate the expected effectiveness of introducing a *least recently used* (LRU) metadata cache for the HDFS.

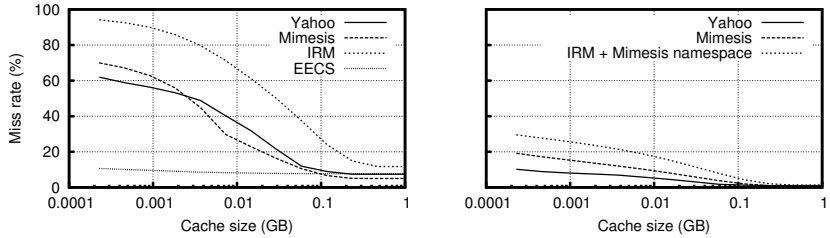
### B. LRU metadata cache

We developed a simulator that replays a metadata trace and calculates the cache miss rate[6] under different eviction policies; we implemented and evaluated a *least recently used* (LRU) policy. Figure 8 shows the miss rate for varying cache sizes[7], calculated after the cache warms up.

To measure the accuracy between the expected cache miss rate (i.e., the one obtained the real Yahoo trace) and the predicted miss rate (i.e., using a model) we use the following metrics: *mean squared error (MSE)*, a classical metric used

[6]The miss rate of a cache is the percentage of accesses for which the data being looked for—in this case, metadata—cannot be located in the cache.

[7]Calculated assuming 1.5 blocks per file and 160 bytes per cache entry, as documented in https://issues.apache.org/jira/browse/HDFS-1114.




| | MSE | RMSE |
|---|---|---|
| *Flat namespace* | | |
| Mimesis | 439.95 | 5.82 |
| IRM | 8939.76 | 26.22 |
| EECS | 12550.90 | 31.07 |
| *Hierarchical namespace* | | |
| Mimesis | 315.06 | 4.92 |
| IRM+Mimesis | 1934.05 | 12.20 |

Fig. 8. LRU metadata cache miss rates for flat (left) and hierarchical (middle) namespaces. The table shows the mean squared error (MSE) and root mean squared error (RMSE) for each model. The synthetic trace created with Mimesis produces the most accurate results (lowest MSE).

in statistical modeling, and *root mean squared error (RMSE)* , which has the same units as the quantity being estimated (in this case, the cache miss rate is expressed as a percentage). Results are shown in Figure 8 (right) and discussed below.

*Case 1:* We consider a LRU cache in which each entry uniquely identifies an object (e.g., using a fully qualified path). Traces that contain flat namespaces, hierarchical namespaces, or unique file identifiers can be used to evaluate this approach.

We evaluate this cache using different 2-hour traces: Yahoo, Mimesis, IRM and EECS. The Yahoo trace constitutes the first two-hours of the original Yahoo (PROD) trace. The Mimesis trace was generated using our model, with the parameters configured using the empirical distributions that describe the Yahoo trace, obtained by the Statistical Analysis Engine.

IRM is a trace generated using interarrivals modeled after the interarrivals of the Yahoo trace, and object accesses sampled from the popularity distribution obtained from the Yahoo trace, assuming the Independent Reference Model (IRM)[8].

We scaled up EECS to match the interarrival rate of Yahoo. To scale up or intensify the trace, we used an approach used by prior studies. The trace is "folded" onto itself as follows: the trace (in this case, EECS) is divided into subtraces, then the subtraces are all aligned to time zero while namespace objects are appended with a unique subtrace identifier. This process increases the number of operations per second (*time*) and the namespace size (*space*). To match the request arrival rate of the Yahoo workload, we folded the EECS trace 18 times (i.e., divided it in 18 subtraces).

Figure 8 (left) shows that using a trace from a system with different access patterns (EECS) is a poor alternative: the EECS miss rate is significantly smaller than the Yahoo one ($RMSE = 31.07\%$) because in EECS objects remain popular longer (see Figure 4), thus leading to a higher hit rate in the cache. The IRM trace provides a slightly better approximation because it was modeled after the original workload ($RMSE = 26.22\%$). The file popularity observed in the Yahoo trace is heavy tailed (see Figure 1), so the IRM trace does have some very popular objects; however, accesses to unpopular objects appear randomly throughout the trace, whereas in Mimesis accesses to unpopular objects tend to appear close together, as given by the AOA distribution which captures the temporal locality of the original trace. As the cache size increases, the behavior of the IRM trace approaches the real trace behavior as a cache with more entries is less sensitive to temporal locality. The best approximation of the results is obtained with Mimesis, having $RMSE = 5.82\%$.

[8]The IRM assumes that object references are statistically independent.

*Case 2:* Consider a metadata cache in which each entry of the cache uniquely identifies an object within a directory; the request to access the file `/a/path/to/a/file` requires one cache lookup for every element in the path. To evaluate this cache, we need a trace that contains information about the fully qualified path to each object (or a mechanism to associate unique object IDs to fully qualified names).

We evaluate the cache miss rate using different traces: Yahoo, Mimesis, and IRM + Mimesis namespace. Yahoo and Mimesis correspond to the same traces described before.

The IRM + Mimesis namespace was generated as follows: we first used Mimesis' Namespace Creation Module to create a namespace modeled after the original Yahoo namespace; this is the same namespace used in the Mimesis trace. We then created a random permutation of the objects in the namespace to eliminate any bias in the order in which the Namespace Creation Module outputs the list of objects in the namespace. Next, we assigned a rank to each object, according to the randomized order: the first object in the list was assigned rank 1, the second object was assigned rank 2, etc. Finally, we used the IRM model to sample objects according to the (ranked) popularity distribution of objects in the Yahoo trace, associating each rank in the sample with one file in the namespace as given by the order of the random permutation.

This cache has a lower miss rate (see Figure 8, middle), resulting from the hits to the directories at the lower depths in the hierarchy tree. Mimesis outperforms the IRM + Mimesis namespace approach ($RMSE$ is $4.92\%$ vs. $12.20\%$) because it is able to capture the temporal locality of the original workload, while the independent reference model does not.

Our results show that our model provides a good approximation to the real workload ($RMSE < 6\%$). We are exploring further improvements, like combining our model with explicit file popularity information, which could help minimize the $MSE$ at the cost of increasing the complexity of the model (and corresponding performance degradation of Mimesis) so the current, simpler, model would be still valuable.

## VI. Discussion and related work

Release of petascale traces by industry would open research opportunities in next-generation storage system design. The first step is obtaining those traces. For some systems, like HDFS, this may be simple since metadata accesses can be logged for auditing purposes and namespace snapshots can be

obtained using existing tools. For other systems, unobtrusive mechanisms to obtain these traces may not be available, and should thus be implemented before the traces can be recorded.

Once obtaining the traces is possible, industry can (a) release anonymized traces, or (b) model the workloads of their traces and release these models. To enable the latter, researchers should come up with expressive metadata workload models and tools to process the traces and obtain the models. Workload generators or compilers can be built to take the models as an input and generate realistic synthetic workloads or configuration files in the languages of existing benchmarking tools. While synthetic workloads will, by definition, differ from the original ones, they are useful if they maintain the characteristics of the original workload that the researcher is interested in and, when used in evaluations, lead to results within some small margin of those that would be obtained with the original workload [19]. Selecting those features that make a workload relevant is crucial to this process [8]. Our model and tools constitute a step towards this goal.

Some tools provide a subset of the features of Mimesis. *mdtest* generates metadata intensive scenarios; however, it does not provide a way to fit the workload to real traces or realistic namespaces. Impressions [4] generates realistic file system images; however, it is not readily coupled with a workload generator to easily reproduce workloads that operate on the generated namespace. Furthermore, the generative model used by Impressions to create the file system hierarchy is not able to reproduce the distributions observed in our analysis.

Fsstats [9] runs through a file system namespace and captures statistics on file attributes, capacity, directory size, file name length and age, etc. The output of fsstats provides empirical CDFs, but details on the shape of the hierarchy tree are limited to the directory size histogram. LANL has released fsstats reports of large namespaces (up to 0.5 PB)[9].

MediSyn [17] captures the temporal locality of (media server) request streams in a way similar to Mimesis; however, it does not capture or reproduce the storage namespace.

ScalaIOTrace [20] compresses traces so that they can easily be shared, but preserves only minimal data access patterns.

FileBench[10] shares some similarities with Mimesis; however, it lacks a method to extract the statistics from real traces and the configurable statistical parameters on which this tool currently operates does not capture the level of detail captured by Mimesis. On the other hand, it has a mature replay implementation suitable for networked file systems.

## VII. Conclusions

We considered the case of evaluating namespace metadata management schemes for next-generation file systems suitable for Big Data workloads, and showed why a common evaluation approach—using old traces from traditional storage systems—is not a good alternative. Big Data storage traces and workload models should be used instead; specifically, a *namespace metadata trace* should contain information about both the namespace and the storage workload atop that namespace. We proposed a statistical model that can capture the relevant properties of the namespace and workload, and developed Mimesis, a system that uses this model to generate synthetic traces that mimic the relevant statistical properties of the original Big Data trace. Through a case study of a LRU namespace metadata cache we showed how the traces generated by Mimesis produce accurate results ($RMSE < 6\%$).

[9]http://institutes.lanl.gov/data/fsstats-data/

[10]http://sourceforge.net/projects/filebench/

## Acknowledgments

We thank Prof. Marianne Winslett for her help in improving this manuscript. This work was partially completed during C. Abad's internship at Yahoo!. R. Campbell and C. Abad are supported in part by AFRL grant FA8750-11-2-0084. Y. Lu is partially supported by NSF grant CNS-1150080. H. Luu is supported by NSF grant 0938064.

## References

[1] Abad, C., Lu, Y., and Campbell, R. DARE: Adaptive data replication for efficient cluster scheduling. In *IEEE CLUSTER* (2011).

[2] Abad, C., Luu, H., Lu, Y., and Campbell, R. Metadata workloads for testing Big storage systems. Tech. rep., UIUC, 2012. http://hdl.handle.net/2142/30013.

[3] Abad, C., Roberts, N., Lu, Y., and Campbell, R. A storage-centric analysis of MapReduce workloads: File popularity, temporal locality and arrival patterns. In *Proc. IEEE IISWC* (2012).

[4] Agrawal, Arpaci-Dusseau, and Arpaci-Dusseau. Generating realistic Impressions for file-system benchmarking. *TOS 5* (2009).

[5] Agrawal, N., Bolosky, W., Douceur, J., and Lorch, J. A five-year study of file-system metadata. *TOS 3* (2007).

[6] Alam, S., El-Harake, H., Howard, K., Stringfellow, N., and Verzelloni, F. Parallel I/O and the metadata wall. In *PDSW* (2011).

[7] Anantharanayanan, G., Ghodsi, A., Wang, A., Borthakur, D., Kandula, S., Shenker, S., and Stoica, I. PACMan: Coordinated memory caching for parallel jobs. In *Usenix NSDI* (2012).

[8] Chen, Y., Srinivasan, K., Goodson, G., and Katz, R. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *SOSP* (2011).

[9] Dayal, S. Characterizing HEC storage systems at rest. Tech. Rep. CMU-PDL-08-109, CMU, 2008.

[10] Devulapalli, A., and Ohio, P. File creation strategies in a distributed metadata file system. In *IPDPS* (2007).

[11] Geist, A. Paving the road to exascale. *SciDAC Review*, 16 (2010).

[12] Leung, Shao, Bisson, Pasupathy, and Miller. Spyglass: Fast, scalable metadata search for large-scale storage systems. In *FAST'09*.

[13] McKusick, K., and Quinlan, S. GFS: Evolution on fast-forward. *Commun. ACM 53* (2010).

[14] Nicolae, B., Antoniu, G., and Bougé, L. Enabling high data throughput in desktop grids through decentralized data and metadata management: The blobseer approach. In *EuroPar09* (2009).

[15] Patil, S., and Gibson, G. Scale and concurrency of GIGA+: File system directories with millions of files. In *Usenix FAST* (2011).

[16] Shvachko. HDFS scalability: Limits to growth. *Usenix ;login* (2010).

[17] Tang, Fu, Cherkasova, and Vahdat. MediSyn: A synthetic streaming media service workload generator. In *NOSSDAV* (2003).

[18] Tarasov, V., Bhanage, S., Zadok, E., and Seltzer, M. Benchmarking file system benchmarking. In *HotOS* (2011).

[19] Tarasov, V., Kumar, K., Ma, J., Hildebrand, D., Povzner, A., Kuenning, G., and Zadok, E. Extracting flexible, replayable models from large block traces. In *Usenix FAST* (2012).

[20] Vijayakumar, K., Mueller, F., Ma, X., and Roth, P. Scalable I/O tracing and analysis. In *PDSW* (2009).

[21] Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., Zelenka, J., and Zhou, B. Scalable performance of the Panasas parallel file system. In *Usenix FAST* (2008).

[22] Zaharia, Borthakur, SenSarma, Elmeleegy, Shenker, and Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys* (2010).