



A Model-Based Namespace Metadata Benchmark for HDFS

Cristina L. Abad, Escuela Superior Politécnica del Litoral; Yi Lu and Roy H. Campbell, University of Illinois at Urbana–Champaign; Nathan Roberts, Yahoo, Inc.

<https://www.usenix.org/conference/icac14/technical-sessions/presentation/abad>

**This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).**

June 18–20, 2014 • Philadelphia, PA

ISBN 978-1-931971-11-9

**Open access to the Proceedings of the
11th International Conference on
Autonomic Computing (ICAC '14)
is sponsored by USENIX.**

A Model-Based Namespace Metadata Benchmark for HDFS

Cristina L. Abad
Escuela Superior Politécnica del Litoral

Yi Lu Roy H. Campbell
University of Illinois, Urbana-Champaign

Nathan Roberts
Yahoo, Inc.

Abstract

Efficient namespace metadata management is increasingly important as next-generation storage systems are designed for peta and exascales. New schemes have been proposed; however, their evaluation has been insufficient due to a lack of an appropriate namespace metadata benchmark. We describe MimesisBench, a novel namespace metadata benchmark for next-generation storage systems, and demonstrate its usefulness through a study of the scalability and performance of the Hadoop Distributed File System (HDFS).

1 Introduction

There are no metadata-intensive benchmarks with realistic workloads for next-generation storage systems [1, 13]. A few existing tools [7, 9] are useful as microbenchmarks, but do not reproduce realistic workloads.

The storage community has long-acknowledged the need for benchmarks based on realistic workloads, and programs like SPECsfs2008 [12] and Filebench [6] are extensively used for this purpose in traditional storage systems. However, emerging Big Data workloads like MapReduce [2] have not been properly synthesized yet.

We present MimesisBench, a novel metadata-intensive storage benchmark suitable for Big Data workloads. MimesisBench consists of a workload modeling tool, a workload generator, and a workload profile from a large cluster at Yahoo. More workloads will be added in the future.

The model on which MimesisBench is based [3] allows it to generate type-aware workloads, in which specific types of file behavior can be isolated or modified for ‘what-if’ and sensitivity analysis. These types of files are modeled autonomically, using unsupervised statistical clustering. The model also supports multidimensional workload scaling.

MimesisBench’s Hadoop-based implementation allows it to be used in any storage system that is compatible with Hadoop (e.g., HDFS, Ceph, CassandraFS, Lustre). We have released the benchmark and workloads as open source so that other researchers can benefit from it¹.

This paper makes two contributions. First, we extend

a model for temporal locality and popularity in object request streams [3] to: (1) include other operations in addition to regular accesses to objects (opens), (2) support pre-existing files (created before the benchmark), and (3) support a realistic hierarchical namespace. Second, we use this model to implement a metadata-intensive storage benchmark to issue realistic workloads on distributed storage systems. This benchmark can be used to evaluate the performance of storage systems without having to deploy a large cluster and its applications.

2 Model

We extend a model we proposed for generating accesses to objects (e.g., opens to files) [3], which is able to reproduce temporal correlations in object request streams that arise from the long-term popularity of the objects and their short-term temporal correlations. This model is suitable for Big Data workloads because it supports highly dynamic populations, it is fast and scalable to millions of objects, and it is workload-agnostic so it can be used to model emerging workloads.

Objects or files in a stationary segment of a request stream are modeled as a set of *delayed renewal processes* [11] (one per object). Each object in the stream is characterized by its time of first access, its access interarrival distribution, and its active span (time during which an object is accessed). With this approach, the system-wide popularity distribution asymptotically emerges through explicit reproduction of the per-object request arrivals and active span [3]. However, this model is unscalable, as it models each object independently.

To reduce the model size, a lightweight version uses unsupervised statistical clustering (k-means) to identify groups of objects with similar behavior and significantly reduce the model space by modeling “types of objects” instead of individual objects. As a result, the clustered model is suitable for synthetic workload generation.

2.1 Extensions to the model

The model described above cannot be used directly to test a storage system since it only reproduces accesses (e.g., opens) to an object (i.e., file) and not other operations that are also critical in a storage system like creates and deletes. We propose the following extensions to the

¹Available: <http://sites.google.com/site/cristinaabad>

original model to make it suitable for namespace metadata benchmarking: (1) additional storage system operations, (2) pre-existing files, and (3) realistic namespaces.

2.1.1 Extension 1: Additional operations

We first extend our model to include file creations, deletions and list operations in addition to regular opens. We focus on these operations because together they constitute more than 95% of the namespace metadata operations in MapReduce clusters [2], thus accounting for the vast majority of the workload.

The **list** operations are like opens and only *read* or access the namespace. Thus, we model both list and opens together as *accesses* to files. A parameter keeps track of the percentage of read operations that constitute opens and those that are list.

On the other hand, **creates** and **deletes** *write* or modify the namespace, and are characterized independently.

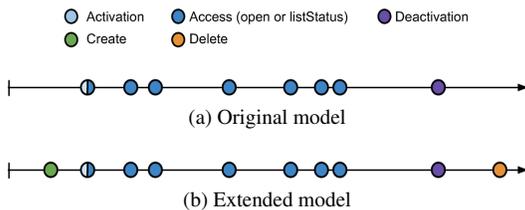


Figure 1: Operations on a single file, with the original and the extended model. The file’s activation time is given by its creation time plus the delay to its first access; the deactivation time is given by its activation time plus its active span.

Figure 1 shows how the original and extended models generate operations on a file of a particular type (cluster). The extended model requires the following additional per-cluster statistical information: distribution of create interarrivals, distribution of delay to deletion (relative to the object’s last access in the stream, which is given by the time of first access + active span), and percentage of accesses that are open operations (list are the remaining percentage). In addition, the activation time is relative to the creation stamp of a file (or to the beginning of the test, for files that already exist at t_0).

Table 1 lists the parameters required by the extended model to generate events to files.

This extension does not affect the access patterns preserved by the original model, which are characterized by the per-cluster access interarrival distribution and the per-cluster distribution of active spans.

2.1.2 Extension 2: Pre-existing files

When benchmarking storage systems, we must consider that their performance depends on the state of the underlying file system [1, 4], namely the pre-existing files and the structure of the hierarchical namespace (the latter is discussed in the next subsection).

We extend our model to keep track of the number of

Table 1: Parameters used to generate events on files; O_i is the object or file i , $i \in \{1, \dots, n\}$; K_j is the cluster or file type j , $j \in \{1, \dots, k\}$.

Symbol	Description
n	number of files in the trace or request stream
k	number of clusters or types of files
F_j	interarrival distribution of accesses to an object in K_j
C_j	interarrival distribution of creations for objects in K_j
G_j	distribution of delay to first access to objects in K_j (relative to creation or t_0); $C_j + G_j = \text{Activation}_j$
H_j	active span distribution $\forall O_i \in K_j$; $\text{Activation}_j + H_j = \text{Deactivation}_j$
D_j	distribution of delay to delete event of objects in K_j ; $\text{Deactivation}_j + D_j = \text{Deletion}_j$
p	percentage of accesses that are opens ($1 - p$: list)
w_j	percentage of objects in K_j ; $\sum_{j=1}^k w_j = 1$
t_{end}	duration of request stream (in milliseconds)
pre_j	number of pre-existing files in K_j
PN	Pre-existing namespace conf. file (generated with NGM)
FD	Array with percentage of files at each depth in hierarchy

files (within each file type) that were created sometime before the beginning of the modeled trace.

We could infer if a file exists prior to the captured trace of namespace events by making the assumption that any file accessed in the trace, but not created during it, is a pre-existing file. However, this approach would lead to an inaccurate model if the trace contains many operations on files that do not exist (e.g., due to users incorrectly entering the name of a file).

To avoid this problem we can use a *namespace metadata trace* that contains a snapshot of the namespace (file and directory hierarchy), in addition to the set of events that operate atop that namespace (e.g., open a file, list directory contents) [1]. The traces we analyzed consist of access logs obtained by parsing the name node audit logs, plus namespace snapshots obtained with Hadoop’s Offline Image Viewer tool.

2.1.3 Extension 3: Realistic namespaces

Earlier, we proposed [1] a statistical model for generating realistic namespace hierarchies, and implemented a *namespace generation module* (NGM) based on it. To the best of our knowledge, this is the only available tool that can generate large realistic namespaces².

Prior to issuing the workload, the NGM is used to generate a realistic directory structure, which preserves the following statistical properties of the original: number of directories, distribution of directories at each depth, and distribution of subdirectories per directory.

To integrate the files to this directory structure, we add a parameter to the model, the *percentage of files at each depth* of the hierarchy, and proportionally assign files to each depth according to this parameter.

²We tested the only other alternative system, the *Impressions* framework [4], and were not able to generate the large namespaces observed in Big Data storage deployments since it was designed to model smaller (more traditional) namespaces. Furthermore, at the time of this writing, the Impressions framework is no longer available for download.

2.2 Assumptions and limitations

Our model can be used to generate realistic synthetic workloads to evaluate the *namespace metadata management subsystems*. Dimensions related to data input/output behavior—like a correlation between file size and popularity or the length of data read/write operations—are out of the scope of our model.

We model **stationary segments** of object request streams. Workloads consisting of a few stationary segments can be divided using the approach in [15]. However, in practice, benchmark runs tend to issue the workload of a period of one hour or less, so the need to consider non-stationary segments is not critical.

We represent arrivals with a sequence of **interarrival times** $X = (X_1, X_2, \dots)$, where X is a sequence of *independent, identically distributed random variables*. In practice, there could be autocorrelations in the arrival process, leading to bursty behavior at different timescales. In [3] we briefly discuss how ON/OFF processes can be used to capture burstiness.

We assume that each arrival process of the accesses to a file in the storage system is **ergodic**, so its interarrival distribution can be deduced from a single realization of the process (i.e., the trace of events representing the segment being modeled). In addition, instead of forcing a fit to a particular interarrival distribution, we use the empirical distribution of the interarrivals inferred from the arrivals observed in the trace being modeled.

Finally, there may be unknown, but important, behavior in the original workloads that our model does not capture. Some of these dimensions, like spatial locality, may be added to our model in the future. However, there is a trade-off of increased complexity due to adding these dimensions.

3 Design

Similar to Hadoop’s DFSIO [16] and S-live [9], MimesisBench is a MapReduce job in which a set of mappers simultaneously issue requests to the storage layer. Each mapper is in charge of issuing the operations on files of a particular type (as encapsulated by the model). The cluster used to run MimesisBench must have at least k nodes available to run a mapper task each, so that the full workload can be issued simultaneously and the mappers do not interfere with each other during their operation.

A run of MimesisBench has two phases: First, the pre-existing files are created; next, the workload is issued.

In each phase, a *job coordinator* parses the parameters and generates configuration files for each worker. In addition, the job coordinator of the first phase creates the hierarchical namespace based on an input parameter file that has been pre-generated with the Namespace Generation Module (NGM).

In the **first phase**, the workers create the target num-

ber of files for each type. A parameter tells the workers to use a flat namespace (create all files in a single, configurable path) or a hierarchical one. Files of each type are created at different levels of the namespace hierarchy, proportionally to the configuration parameter of *files at each depth* (which indicates what percentage of files are located at each depth). The subdirectories at each depth are assigned to a file type, proportionally to the weight of the cluster (w_j) to which the file belongs.

In the **second phase**, the workers issue the load. Each *load generator* worker reads the configuration for the specific file type that it has been assigned and waits in a time-based barrier³ to start issuing the load corresponding to the files that belong to the type it is in charge of. Two data structures are used to keep track of the files and events: a `PriorityBlockingQueue` of files (sorted by the timestamp of the next event—create, open, etc.—of that file) and a `FIFO BlockingQueue` of events to be issued⁴. Three threads coordinate access to these data structures: a file introduction thread adds files to the priority queue (using the cluster’s create interarrival distribution)⁵, another thread continuously polls the priority queue and adds details of the next event to be issued to the back of the FIFO queue. Finally, a consumer thread pulls the information of the next event to be issued from the FIFO queue and schedules it to be issued at the proper time, using Java’s `ScheduledExecutorService`.

All file create operations create files of size zero. This allows us to ignore the effect of writing bytes to a file, handled by the data nodes, and concentrate on evaluating the namespace metadata server (name node in HDFS) performance.

A configurable maximum allowed drift is used to abort a run of the benchmark if the events are falling behind from their original schedule. In that case, more mappers would be needed to issue the workload.

A collector reducer task gathers the stats from the workers and generates aggregate results (see Table 2).

Table 2: Statistics reported by MimesisBench.

Description	Unit
Throughput	<i>ops/sec</i>
Active time (workers)	<i>msecs</i>
Operations issued	<i>ops</i>
Successful creates/deletes/opens/list	<i>ops</i>
Average latency: create/delete/open/list	<i>msecs</i>

3.1 Scaling workloads

A workload can be scaled across several dimensions:

³The clocks of the nodes in a Hadoop cluster are typically synchronized to support Kerberos authentication.

⁴The size of these queues in the current implementation is set to be the number of files of that particular type for the former, and 1 500 000 for the latter.

⁵Pre-existing files of a type are added to the queue upon loading.

Number of files. A workload profile comes with a configured number of files, n , as observed in the original trace on which the model is based. One can increase or decrease the number of files to emulate a larger load.

Number of files of a particular type. The user can increase the number of files of only one particular type by increasing n and doing a transformation on the w_j weights. For example, to double the number of objects of type 1 while keeping the number of objects of types 2 to k unchanged, we can obtain the values of n_{new} and $\{w_{1_{\text{new}}}, w_{2_{\text{new}}}, \dots, w_{k_{\text{new}}}\}$ by solving:

$$n_{\text{new}} = n + w_1 \times n \quad (1)$$

$$w_{1_{\text{new}}} \times n_{\text{new}} = 2(w_1 \times n) \quad (2)$$

$$w_{j_{\text{new}}} \times n_{\text{new}} = w_j \times n, \quad j \in \{2, \dots, k\} \quad (3)$$

Time. Interarrivals can be *accelerated* by multiplying the random variable by a scaling factor between 0 and 1. A scaling factor of 1 reproduces the original workload, while a scaling factor of 0 provides maximum stress on the system by issuing all the operations as fast as possible (0 millisecond wait between operations). Interarrivals can also be slowed down by multiplying the random variable by a constant greater than 1.

Active span. The active span random variable can also be multiplied by a user-defined constant. Modifying the active span has the effect of modifying the number of accesses of the files, thus modifying their popularity.

3.2 Isolating workloads

The user can choose to isolate the workload of a particular file type or cluster (i.e., turn-off the other types of files). This can be used to analyze how a particular type of file affects the performance of the system.

Summary Table 3 shows a summary of the features of MimesisBench and other related tools.

4 Evaluating a Big Data storage system

We demonstrate the usefulness of MimesisBench by using it to evaluate the performance and scalability of the HDFS name node across several dimensions.

We modeled a 1-day (12/1/2011) *namespace metadata trace* from a Hadoop cluster at Yahoo. The trace came from a 4100-node production cluster [10]. It contains 60.9 million opens events that target 4.3 million distinct files, and 4 PB of used storage space. For a detailed workload characterization of this cluster see [2]. We modeled the trace using 30 file types or clusters (i.e., $k = 30$ for the k-means clustering algorithm). We chose this value of k because it was the smallest for which we could obtain a close approximation of the file popularity

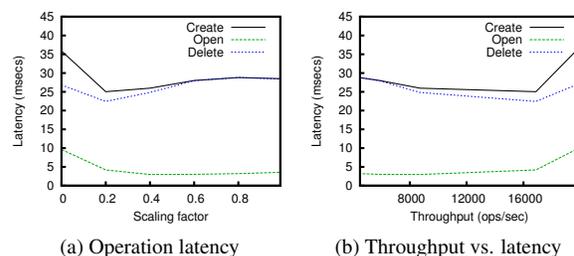


Figure 2: *Left:* Operation latency for varying interarrival speeds, averaged across five runs. In each run, the interarrival random variable, X , was multiplied by a constant, c , shown in the x-axis. When $c = 0$, the operations are issued as fast as possible. When $c = 1$ the operation interarrival mimics the interarrivals of the original trace. For all data points, the standard deviation is very small (< 0.7 msecs). *Right:* Throughput vs. operation (open) latency.

distribution (Kolmogorov-Smirnov distance between the real and synthetic CDFs $< 4\%$).

Our performance testbed had 32 nodes, each with 2 Xeon 2.4GHz quad-core processors and 24GB RAM.

4.1 Latency of opens, creates and deletes

Figure 2 shows the effect on the operation latency as interarrivals are accelerated. In general, read operations are faster than write operations because acquiring an exclusive lock is not necessary to read the namespace.

The latency of the operations is not initially affected by issuing them faster. This shows, that the performance of the name node is not degraded as more operations are issued per second. However, when the clients issue more than 16,800 operations per second ($c \geq 0.2$), the name node's performance starts degrading rapidly. This information can be used to determine whether the name node can properly support an increased workload in the future.

4.2 Flat versus hierarchical namespaces

Figure 3 shows the impact a hierarchical namespace (versus a flat one) has on the name node. The performance degrades significantly faster on a flat namespace than on a hierarchical one. The hierarchical namespace can serve up to 19,696 ops/sec versus 10,284 ops/sec for the flat namespace.

These results show that using benchmarks that create files in a flat namespace (see Table 3) is not desirable as they place a heavier and unrealistic burden on the locking mechanisms of the metadata server. In this case, the problem is with the locking used to log namespace write operations used for auditing purposes.

4.3 Isolating workloads

We isolated the workload of two file types with different access patterns and observe the effects on latency and throughput (see Table 4). We chose these two clusters because they represent two extreme, read-mostly (cluster 29) and write-heavy (cluster 17), yet realistic, workloads.

We ran several tests with events issued at normal speed

Table 3: Features of MimesisBench and other related tools. See Section 5 for a description of these tools.

Feature	NNBench	DFSIO	S-live	Filebench	SPECsfs	mdtest	MimesisBench
For next-generation storage	✓	✓	✓				✓
Metadata-intensive	✓					✓	✓
Realistic workloads				✓	✓		✓
Type-aware workload				✓			✓
Autonomic type-awareness							✓
Hierarchical namespace	Semi-flat ⁺		Semi-flat ⁺	Limited*	Fixed		✓
Issues I/O load		✓	✓	✓	✓		✓

⁺ Only multiple directories at depth 1 are supported.

* Creates hierarchies with a given depth and width. Characteristics like subdirectories per directory and directories per depth, are not supported.

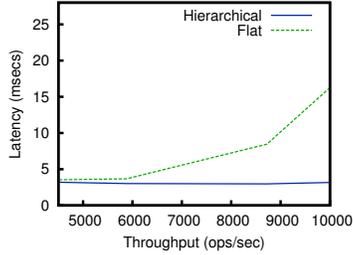


Figure 3: Throughput vs. open latency in a hierarchical and a flat namespace. Five runs; standard deviation < 1.5 msecs.

(interarrival scaling factor = 1) and at full speed (interarrival scaling factor = 0). To ensure a fair comparison, we adjusted the number of mappers issuing the workload so that the total number of operations issued in both tests was roughly the same. At 30 mappers for cluster 17, and 12 mappers for cluster 29, the number of operations issued in the tests was $3559101 \pm 0.9\%$.

Figure 4 shows the latency of open events. The latency of the open events degrades significantly more in a write-heavy workload: When operations are issued at full speed, the latency of opens in the write-heavy workload increases 3.9x in cluster 17 versus 1.4x in cluster 29. In addition, at maximum issuing speed (interarrival scaling = 0) the name node can serve 8 times more operations when the workload constitutes only reads: 53,233 vs. 6,453 ops/sec for clusters 29 and 17, respectively.

Table 4: Characteristics of the two clusters whose workload was isolated. We chose these two clusters because they represent two extreme, read-mostly and write-heavy, workloads.

	Cluster 17	Cluster 29
Mean interarrivals (regular accesses)	179.61 msecs	4.92 msecs
Mean creates interarrivals	40.64 msecs	87,355.00 msecs
Mean active span	3.33 mins	8.33 mins
Percentage of read operations	69%	≈ 100%
Percentage of write operations	31%	≈ 0%

4.4 Evaluating a proposed enhancement

A common use for benchmarks is to evaluate a new design against an old design. In this subsection, we show the results of one example of this type of evaluation.

The HDFS-5239 [8] Jira allows the namespace lock fairness to be changed from fair (default) to unfair. This

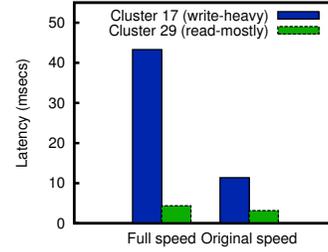


Figure 4: Latency of open events, with two interarrival scaling factors (0 or operations at full speed, and 1 or operations issued at their original speed), averaged across five runs, for two clusters with different read/write ratios (see Table 4). Standard deviation < 0.5 msecs.

change may be desirable in large clusters, as an unfair lock has a higher throughput than a fair lock.

Table 5: Throughput improvement obtained when using an unfair namespace lock (HDFS-5239), for four different workloads.

	Fair lock	Unfair lock	Improvement
Hierarchical	19 696 ops/sec	21 899 ops/sec	11%
Flat	10 255 ops/sec	11 299 ops/sec	10%
Read-mostly	53 324 ops/sec	103 331 ops/sec	94%
Write-heavy	8 694 ops/sec	8 701 ops/sec	1%

Table 5 shows the name node throughput improvement for the following scenarios: full workload on a hierarchical namespace, full workload on a flat namespace, a realistic read-mostly workload (cluster 29), and a realistic write-heavy workload (cluster 17). The improvement when the full workload is issued is around 10%. However, for a read-mostly workload, the improvement is as high as 94%. This is a result of the bottleneck that exists in the logging mechanism used for write operations.

4.5 Stability of the results

Benchmarks are often used to evaluate new designs against an old design, or against another competing new design. An improvement of 10% in performance may be desirable to push in a large system, as long as we can trust the results of the benchmark. For this reason, it is important to have a small variability in the results produced by a benchmark. For example, it is not reasonable to trust a 10% performance improvement if the coefficient of variation, c_v , of the results is 8%.

Our experiments had a very small standard deviation and corresponding coefficient of variation (or the ratio

of the standard deviation to the mean). For all the sets of experiments we ran, the greatest observed coefficient of variation was 2.38%, with an average coefficient of variation of 1.08%. These results suggest that MimesisBench is suitable as a tool to evaluate expected performance improvements of new designs.

5 Related work

Some prior tools provide a subset of the features of MimesisBench, as summarized in Table 3.

Mdtest issues metadata intensive workloads. However, it does not support realistic workloads or namespaces. In addition, *mdtest* interfaces with the storage system via system calls and has not been ported to the interfaces of next-generation storage systems, nor does it allow distributed workload generation.

Filebench [6] uses a POSIX-compliant interface and includes traditional workloads like web server, file server, and database server. Emerging Big Data workloads have not been included as pre-defined workloads.

SPECsfs2008 [12] is used as a standard to enable comparison of file server throughput and response times across different vendors and configurations. It supports NFSv3 and CIFS APIs and comes with a pre-defined workload based on enterprise NFS and CIFS workloads.

For HDFS, the Hadoop developer community has designed two benchmarks that can test I/O operations and do simple stress-tests on storage layer: *DFSIO* [16] and *S-live* [9]. However, these benchmarks do not reproduce realistic workloads and can only be used as microbenchmarks. Another Apache tool, *NNBench* [7] was created to benchmark the HDFS name node; however, it can only issue one type of operation at a time, and does not work atop a realistic namespace.

Tarasov et al. [14] found that traditional workloads, like those provided with *Filebench*, are a poor replacement for virtual machine workloads. We consider another emerging workload: MapReduce clusters.

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, nor is it able to scale to the large hierarchies observed in the Big Data systems we have studied.

Chen et al. [5] proposed the use of multi-dimensional statistical correlation (k-means) to obtain storage system access patterns and design insights in user, application, file, and directory levels. However, the clustering was not leveraged for workload generation or benchmarking.

In earlier work, we developed *Mimesis* [1], a synthetic trace generator for namespace metadata traces. However, it was too CPU-intensive to issue operations at real-time

and as such is inapplicable for benchmarking real systems (though its synthetic traces could be used in trace-based evaluations). In addition, its model does not reproduce file popularity.

6 Conclusions

We presented *MimesisBench*, a metadata-intensive storage benchmark suitable for Big Data workloads. *MimesisBench* consists of a workload-generating software and a workload from a Yahoo Big Data cluster.

MimesisBench is extensible and more workloads can be added in the future. It is based on a novel model that allows it to generate type-aware workloads, in which specific type of file behavior can be isolated or modified for ‘what-if’ and sensitivity analysis. In addition, it supports multi-dimensional workload scaling.

MimesisBench is implemented on top of the Apache Hadoop framework, which allows it to be used in any storage system that is compatible with Hadoop. We have released the benchmark and workload as open source.

A study of the performance and scalability of HDFS was presented to show the usefulness of metadata-intensive benchmarking using *MimesisBench*.

Acknowledgments

We thank our anonymous reviewers for their many insightful comments and suggestions. This work was partially completed during C. Abad’s PhD studies at UIUC and during her internship at Yahoo. R. Campbell and C. Abad were supported in part by AFRL grant FA8750-11-2-0084. Y. Lu is partially supported by NSF grant CNS-1150080.

Bibliography

- [1] ABAD, C. L., LUU, H., ROBERTS, N., LEE, K., LU, Y., AND CAMPBELL, R. H. Metadata traces and workload models for evaluating Big storage systems. In *Proc. UCC* (2012).
- [2] ABAD, C. L., ROBERTS, N., LU, Y., AND CAMPBELL, R. H. A storage-centric analysis of MapReduce workloads: File popularity, temporal locality and arrival patterns. In *Proc. IISWC* (2012).
- [3] ABAD, C. L., YUAN, M., CAI, C., ROBERTS, N., LU, Y., AND CAMPBELL, R. H. Generating request streams on Big Data using clustered renewal processes. *Perf. Eval.* 70, 10 (Oct. 2013).
- [4] AGRAWAL, ARPACI-DUSSEAU, AND ARPACI-DUSSEAU. Generating realistic Impressions for file-system benchmarking. *ACM TOS* 5 (2009).
- [5] CHEN, Y., SRINIVASAN, K., GOODSON, G., AND KATZ, R. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proc. SOSP* (2011).

- [6] FileBench. sourceforge.net/projects/filebench, Mar. 2013. Last accessed: April 15, 2013.
- [7] NOLL, M. G. Benchmarking and stress testing an Hadoop cluster with TeraSort, TestDFSIO & co. www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench, Apr. 2011. Last accessed: January 18, 2014.
- [8] SHARP, D. Allow FSNamesystem lock fairness to be configurable. issues.apache.org/jira/browse/HDFS-5239, 2013. Last accessed: March 26, 2014.
- [9] SHVACHKO, K. A stress-test tool for HDFS (Apache JIRA HDFS-708). issues.apache.org/jira/browse/HDFS-708, 2010. Last accessed: December 30, 2013.
- [10] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The Hadoop Distributed File System. In *Proc. MSST* (2010).
- [11] SIEGRIST, K. Chapter 14: Renewal processes. In *Virtual Laboratories in Probability and Statistics*. 2013. Last accessed: March 23, 2013.
- [12] SPECsfs2008. <http://www.spec.org/sfs2008/>, Apr. 2013. Last accessed: December 30, 2013.
- [13] TARASOV, V., BHANAGE, S., ZADOK, E., AND SELTZER, M. Benchmarking file system benchmarking. In *HotOS* (2011).
- [14] TARASOV, V., HILDEBRAND, D., KUENNING, G., AND ZADOK, E. Virtual machine workloads: The case for new benchmarks for NAS. In *Proc. Usenix FAST* (2013).
- [15] TARASOV, V., KUMAR, S., MA, J., HILDEBRAND, D., POVZNER, A., KUENNING, G., AND ZADOK, E. Extracting flexible, replayable models from large block traces. In *Proc. Usenix FAST* (2012).
- [16] VAVILAPALLI, V. K. Delivering on Hadoop .Next: Benchmarking performance. hortonworks.com/blog/delivering-on-hadoop-next-benchmarking-performance, 2012. Last accessed: December 30, 2013.