File System Trace Replay Methods Through the Lens of Metrology

Thiago Emmanuel Pereira*, Francisco Brasileiro† and Livia Sampaio‡
Department of Computing and Systems, Universidade Federal de Campina Grande
Campina Grande, Brazil
Email: *temmanuel@copin.ufcg.edu.br, †fubica@computacao.ufcg.edu.br, ‡livia@computacao.ufcg.edu.br

***Abstract*—There are various methods to evaluate the performance of file systems through the replay of file system traces. Despite this diversity, little attention was given on comparing the alternatives, thus bringing some skepticism about the results attained using these methods. In this paper, to fill this understanding gap, we analyze two popular trace replay methods through the lens of metrology. This case study indicates that the evaluated methods provide similar, good precision but are biased in some scenarios. Our results identified limitations in the implementation of the replay tool as well as flaws in the established practices to experiment with trace replayers as the root causes of the measurement bias. After improving the implementation of the trace replayer and discarding inappropriate experimental practices, we were able to reduce the bias, leading to lower measurement uncertainty. Finally, our case study also shows that, in some cases, collecting only the file system activity is not enough to accurately replay the traces; in these cases, collecting resource consumption information, such as the amount of allocated memory, can improve the quality of trace replay methods.**

I. Introduction

Performance evaluation has, for many years, helped the adoption, development, and operation of file systems. In this paper, we focus on performance evaluation based on the replay of file system activity traces. Compared to other performance evaluation methods, such as analytical models [1], [2], simulations [3], [4], and micro and macro benchmarks [5], trace-based performance evaluation is believed to provide the best representation of real workloads.

There are several methods for the replay of file system traces. The main differences between them lie on the code architecture followed by the replayer — compilation-based or event-based — and on the replay model — open or closed [6]. Compilation-based replayers [7], [8] create the source code of a program that emulates the activity captured in the trace: the workload is generated by executing this program. Event-based replayers [9], [10] are generic programs able to replay any arbitrary trace given as input. Regarding the replay models, in closed ones, the arrival of new requests may depend on the completion of the previous requests while in open ones this is not the case.

Given this diversity, how can one choose the best tool for one's purpose? To answer this question, it is instrumental to assess the quality of trace replay tools. These methodological aspects have been neglected, thus bringing some skepticism to the results attained using trace-based performance evaluation [11], [12].

In a recent work, we have drawn upon metrology to analyze the quality of trace capture methods [13]. We found that trace capture shows significant bias in some cases. Fortunately, the bias can be reduced by applying a calibration procedure. We also found that trace capture can be affected by background activity in the experimental environment. As a consequence, gathering information about the background activity is crucial for appropriate bias correction. In this paper, we apply the same metrology framework to improve the understanding of the replay of file system traces. To the best of our knowledge, this comparison of the design of trace replayers was not considered so far.

In our case study we adopted the `ARTC` trace replayer as the compilation-based alternative [7], while we developed an event-based trace replayer following the `TBBT` design [9] and established practice on the implementation of such systems [8], [14].

We captured and replayed two workloads of increasing complexity levels. The first workload is generated by a microbenchmark program that performs a sequence of file system related system calls. The second, more complex, workload is generated by the filebench [15] file system benchmark which was configured to emulate the I/O activity of a file-server.

Our evaluation metric is the file system response time as perceived by the applications. In our analysis of response time, we are interested in assessing some important characteristics of the trace replay methods. These characteristics include precision, bias, and uncertainty. Precision quantifies the closeness of agreement between measured values obtained by replicated trace replay experiments. Bias is the closeness of agreement between the measured values obtained with a trace replay method and a conventional reference value, obtained. The precision and the bias found empirically are combined in a single metric, the uncertainty (Section III-F), which characterizes the dispersion of the measured values within a interval with a confidence level.

However simpler than real workloads, the workloads generated by the microbenchmark and macrobenchmark programs allow us to identify trace replay characteristics and limitations. For example, the replay of the first workload indicates that the two methods that we evaluated are precise; they show equivalent variability with the workload they replay. On the other hand, we found that the trace replay methods have different biases, with the event-based method showing higher bias than the compilation-based one. For instance, the uncertainty to replay sequential read workloads, at the 95.5% confidence interval, was up to 253% for the event-based replayer, while for the compilation-based replayer it was no more than 20.7% .

We used the results of the bias analysis to identify the sources of trace replay measurement errors. It turns out that established practice on the design of such systems has flaws that have been highlighted by our analysis. Some of these error sources were mitigated by changes in the implementation of the event-based replayer. After the improvements, the uncertainty to replay sequential read workloads was reduced from 253% to no more than 32.9%.

We also found that the mechanisms usually adopted by the replayer tools to determine the duration of time intervals are inappropriate to accurately replay traces with delays between requests. For example, the uncertainty to replay *random write* workloads without pause between requests is no more than 10.9%, while the uncertainty to replay *random write* workloads with pause is up to 22.5%.

The replay of the second workload indicates that the trace replay methods are sensitive to divergences between the capture environment and the replay environment. We found that the replay tools have a smaller memory footprint than the traced program, thus increasing the amount of memory available to the page cache. As a consequence, in the replay environment, the requests to the file system are more likely to be served by the memory subsystem instead of by the disk.

Due to this effect, the uncertainty to replay the second workload is higher than the uncertainty to replay the microbenchmark workload. For instance, for the compilation-based replayer, the uncertainty to replay *read* and *write* operations of the second workload are up to 92.72% and 33.79%, respectively, while the uncertainty to replay *read* and *write* operations of the first workload are no more than 20.7% and 1.3%, respectively. For the event-based replayer, the uncertainty to replay *read* and *write* operations of the second workload are up to 118.27% and 79.81%, respectively, while the uncertainty to replay *read* and *write* operations of the first workload are no more than 32.9% and 1.6%, respectively.

Before presenting the case study, we present the general aspects of metrology, including some metrology terminology adopted and a method validation protocol (Section II). Then, we describe how to apply the validation protocol in our case study (Section III), and report the results of the validation, in terms of precision, bias and uncertainty (Section IV). We conclude the paper with recommendations drawn from our results (Section V), a review of the related literature contextualizing the contributions of this paper (Section VI), and a discussion of some open problems that we did not consider in our case study (Section VII).

II. Background

Metrology deals with the general problem of measuring with imperfect instruments and procedures [16]. Since the measurement is always imperfect, metrology provides methods for estimating measurement errors and uncertainties. Accurate determination of time intervals is key to trace replay tools, otherwise they are not able to emulate the delays between trace capture requests properly. A possible source of errors arises from the interference between the measurement instrument and the measured objects, since they share the same computer resources.

In this paper, we are interested in *method validation* or *method testing* [17] — the branch of metrology responsible for assessing the fitness to the purpose of a chosen measurement method. It is specially important when there is no standard measurement method established; this is the case for trace replay methods, as we discussed in the Introduction.

In this section, we review a protocol, widely applied in other sciences, to conduct experimental method validations. This protocol accounts for the measurement errors likely to be controlled within a single laboratory. The protocol consists of the following steps: i) measurand definition; ii) measurement procedure specification; iii) uncertainty sources identification; iv) measurement characterization; v) calibration analysis; and vi) determination of measurement uncertainty.

In the first two steps, we consider the careful the definition of **measurand**[1], i.e. the quantity intended to be measured, and the **measurement method**, which specifies how to use a **measurement instrument** to obtain the value of the measurand and the measurement conditions (the environment) under which the measurements must be conducted.

In the *Uncertainty sources identification* step, the possible sources of uncertainty and typical limitations of the chosen measurement instruments are pinpointed. In this part of the protocol, unknown factors may be overlooked, and therefore measurement uncertainty may be underestimated.

In the *Method characterization* step, measurement experiments are conducted to analyze the quality of the measurement methods and instruments. These experiments consider the effects of the source of uncertainty pinpointed in the previous step, and are usually analyzed in terms of precision and bias. These metrics quantify the impact of two types of errors: **random** and **systematic**. A random error is equally likely to underestimate or overestimate the true value of the measurand in repeated measurements.

[1]Bold terms are metrology jargon found in the "international vocabulary of basic and general terms in metrology (VIM)" [18].

On the other hand, a systematic error always goes in the same direction, underestimating or overestimating the true value of the measurand.

In the *Calibration analysis* step, the preliminary bias results are analyzed. When the bias is significant, the bias analysis is used to a calibration procedure which can be latter applied to correct systematic errors.

Finally, the **accuracy** of the measurement method is quantified in the *Measurement uncertainty* step based on the results of the experiments conducted in the *Measurement characterization* step. This is done after the calibration has been performed, if necessary and possible, in the *Calibration analysis* step, in order to correct systematic errors. The results of this last step facilitate deciding whether the method fits within its purpose after correction or there is a need to develop another one (or to relax some of the requirements).

III. Trace replayers validation

In this section, we present how the classic validation protocol described in the previous section can be used to evaluate trace replay methods. We devote a subsection to each of the protocol steps.

A. Measurand specification

Our measurand is the file system response time. This measurand is considered in a sequence of events T, related to the execution of the file system functions, generated by the requests of the trace replayer. The event includes timing information and data such as arguments and returned values from the replayed function.

Thus, each event $t_k \in T$ is a tuple $\langle f_k, b_k, e_k, c_k \rangle$, representing the replay of file system function $f_k \in F$, from the set of file systems functions F, where b_k and e_k are the timestamps just before and after the replay of the function, and c_k is the relevant arguments and returned values.

For each event $t_k = \langle f_k, b_k, e_k, c_k \rangle$ of the sequence of events T, the response time of event t_k is given by the time interval that elapses between the execution of the last and first statements of f_k, i.e. $e_k - b_k$.

B. Measurement procedure definition

In this section we consider the definitions of the methods used to replay traces. These definitions include the measurement conditions (the environment) under which the measurements must be conducted, the measurement instruments (trace replay tools), and the measurement procedures to use these instruments

1) Experimental environment: We conducted the measurements in an Intel E6550 Core 2 Duo 2.33GHz workstation, with 2 GB of main memory, running the Linux 2.6.32-41 kernel. This machine has two hard disks: a 7200 RPM SATA disk with a 8 MB cache, and a 5400 RPM SATA disk with a 32 MB cache. We instrumented an ext4 file system, mounted in the second hard drive.

2) Measurement instruments: In this case study, we adopted compilation-based and event-based trace replay instruments. To experiment with the compilation-based design, we adopted the `ARTC` trace replayer [7]. In our experiments, we applied the `ARTC` without modification. To experiment with the event-based design, we developed a trace replayer based on `TBBT` design [9]; we had to develop a new replayer because there was no public available event-based tool, however, we did that following established practice [8], [14], as will be discussed shortly.

The trace replayer function is adjusted by two important controls: the ordering and the timing policies. An ordering policy defines the dependence relation that any two requests may have, which defines an order in which these requests should be replayed. A timing policy defines the exact time at which the requests need to be replayed, when processing the trace. These controls define the open or closed nature of the trace replay model, as defined in the Introduction.

The `TBBT` and the `ARTC` replayers define two equivalent classes of order policies. The first class (composed by the *ROOT* policy in the `ARTC` replayer and by the *FS dependency* policy in the `TBBT` replayer) leverages the semantics of file system operations to define the order relation between requests. For example, a request to write to a file cannot be replayed before the request that creates that file. The second class (composed by the *temporal* policy in the `ARTC` replayer and by the *conservative* policy in the `TBBT` replayer) replays requests according to the order found in the trace, that is, a request is replayed only after the previous requests (based on the request timestamps) have been replayed.

Since the `TBBT` and `ARTC` order policies are equivalent, for ease the reading of the following sections, we refer to the policies from the first class (*ROOT* policy in the `ARTC` replayer and the *FS dependency* policy in the `TBBT` replayer) as *FS*, and to the policies from the second class (*temporal* policy in the `ARTC` replayer and the *conservative* policy in the `TBBT` replayer) as *temporal*.

The `TBBT` and the `ARTC` replayers also have similar timing policies. Both replayers implemented a *fullspeed* timing policy. In this policy, any possible feedback between requests is ignored and the requests are replayed as fast as possible, as long as they respect the order policy. In addition to the *fullspeed* policy, the `TBBT` replayer implements an additional policy, called *timestamp*. In this policy, the requests are replayed as close as possible to the traced timestamps, again, respecting the order policy.

The compilation and the event-based replayers workloads were generated based on the same traced data. Table I describes the format of the trace.

The `ARTC` replayer, as a compilation-based alternative, takes the captured trace as input and generates a source code equivalent to the captured workload. The generated source code includes a thread for each thread found in the trace. The code statically defines the data associated

Table I: Captured trace format.

Trace field	Description
pid	The id of the process that called the file system function.
tid	The id of the thread that called the file system function.
begin	The timestamp for the time when the file system function was called.
end	The timestamp for the time when the file system function returned.
function	The name of the file system function called.
args	The arguments of the file system function called.
rvalue	The returned value of the file system function called.

to the requests, such as the arguments of file system functions. The source code also defines the sequence of requests that each thread will replay — again, as found in the trace — and the dependency relation specified by the selected order policy. To implement this order relation, the representation of a request in the generated program keeps track of the identification of the dependent requests. At runtime, before executing a request, a thread will first check if the predecessors requests were already completed, blocking on a condition variable when it is not the case.

Our event-based implementation is based on the TBBT design [9]. It avoids the code generation and compilations steps. It takes the captured trace and formats it to a more structured input file. This new structure includes the order relation defined by the selected order policy. The replayer parses this formatted file, and translates the parsed requests to replayed systems calls. These operations are performed by two groups of threads: coordinator and workers. The coordinator thread generates events to be replayed by the worker threads.

Event generation evolves according to the rules defined by the selected order and timing policies and by the progress made by the worker threads. To generate the events, the replayer uses a color marking algorithm shown in Figure 1. In this algorithm, a request has three possible states: i) *unavailable*; ii) *available*; and iii) *replayed.* The coordinator thread generates events to each *available* request and add these events to an *available queue.* A free worker thread takes an event from this queue and executes the related file system function. After the execution, the worker thread adds the event into a *replayed queue.* The coordinator thread takes the event from the queue and changes the request state from *available* to *replayed.* Following, the coordinator changes the state of the successor requests, according to the order relation, from unavailable to available.

Our event-based tool runs as a real time process to avoid the preemption of the replayer process by other processes

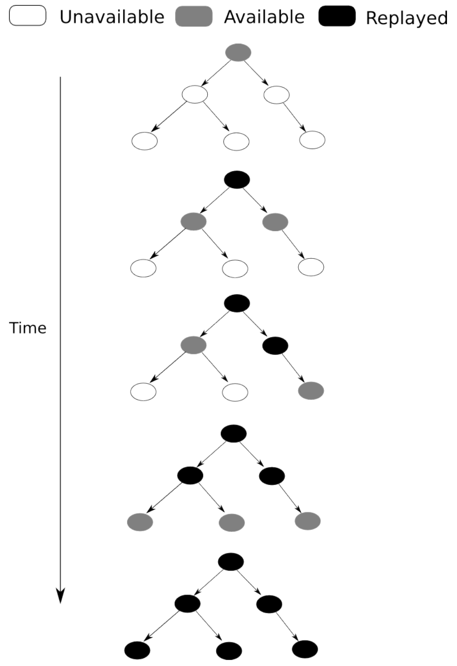

Figure 1: The event-based replayer uses a color marking algorithm to produce events to be replayed by the worker threads. A free worker thread executes available events (grey). After execution, the coordinator marks the event as replayed (black) and marks its unavailable (white) successors as available (grey).

running in the experimental environment [8], [14] [2].

3) Measurement procedures: Before we can replay the trace, we have to recreate the file system state in such way that it reflects the file system content at trace capture time. To recreate the file system state, we collected a snapshot of the file system just before we gathered the trace. After recreating the file system state, the measurement procedure starts by flushing page, *dentry* and *inode* caches. This is to ensure that changes in the file system cache, after a trace replay measurement, do not affect subsequent experiments. After cache flushing, we start the trace replayer execution, wait for its termination and gather the output generated by the trace replayer. To reduce the interference between the storage of the data generated by the trace replay tool and the workload requests, both replayer input and output data trace are stored in a different file system mounted in a different disk.

C. *Uncertainty sources identification*

In this section we describe the most important uncertainty sources for trace replay methods, shown in Figure 2. As we stated in Section II, the identification of uncertainty sources is important because when uncertainty factors are neglected, the measurement uncertainty may be underestimated.

[2]The material necessary to reproduce the results is released in the https://github.com/thiagomanel/mass2016 web repository

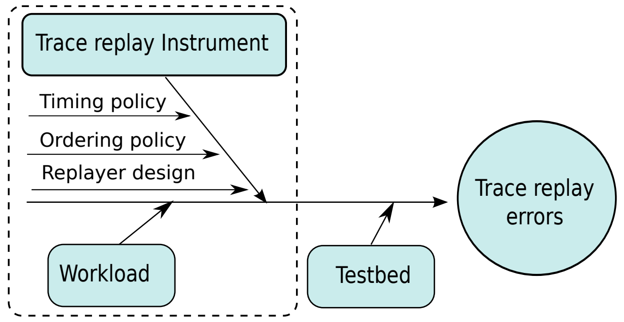

Figure 2: Uncertainty sources that contribute to trace replay measurement errors. In our within-laboratory evaluation, we consider workload and trace replayer instrument sources.

The *workload* impacts the behavior of the file systems and of the operating system. For example, file system flushes are governed by the number of dirty and free pages (both factors depend on the workload) [19]. Since the trace replay instruments, as user processes, are not isolated from the file system and the operating system — for instance, they share the machine's CPU and memory — the workload may also impact trace replay instruments, thus affecting measurement errors.

A perfect *trace replay instrument* would be able to reproduce a trace in such way that the measured values (in our case, the file system response time) equals the ones of the original traced applications running in the reproduction environment.

Naturally, the pitfalls on the design and implementation of the trace replayers undermine building an ideal measurement instrument. For example, in both the event and compilation-based replayers the concurrent access from the replayer threads to critical regions has to be controlled; the delays to control concurrent accesses are sources of systematic errors. In the event-based replayer, since we employ an extra component to coordinate the replay, the systematic errors to control concurrent accesses are likely to be even higher.

The replayer policies are also sources of errors. This is because they shape the replayed workload, for example, by modifying the concurrency level of the workload, and the workload itself is a source of error, as previously described.

Finally, the testbed, composed of the hardware and the system software, also impacts the measurement uncertainty. For instance, the file system acts as a software layer between the applications and the input/output system. This interaction with the underlying system software layers defines the file system performance. This uncertainty source is particular important when one analyzes reproducibility issues, for example, when measurements from different laboratories are compared. In this paper, however, since we evaluate the uncertainty sources that affect measurements made within a single laboratory, we not consider the testbed influence.

D. *Measurement characterization*

To characterize measurement, measured values are compared to the true value of the measurand. As we described in Section II, since this true value is unknown, reference values are adopted as approximations to them.

There are two methods widely used to obtain reference values: i) to measure with a reference method (a method with known, lower errors); and ii) to measure reference objects (an object with known measurand values, such as standard-weights).

We cannot apply the first method, since there is no low-error trace replayer to be applied as a reference method. For this reason, we adopted the second method, based on reference objects.

We adopted as reference values the measured file system response time as shown in the captured trace. Note that the capture method may also be a source of uncertainty. To mitigate this problem, instead of adopting popular trace capture tools such `strace` and `SystemTap`, we collected the execution timestamps at the application level (calls to the unix `clock_gettime` function were added to the traced program). This approach gives the closest measurement of file system response time, as perceived by the applications, thus closer to the true value of the measurand.

In the characterization, we traced the execution of two programs which generate workloads of increasing levels of complexity. The first program is a microbenchmark that generates file system related system calls at an specific load level. The second program is the filebench [15], a file system benchmark that we configured to generate a file-server workload.

The load level of the microbenchmark is given by the number of worker threads performing file system requests. Each thread performs a configurable number of file system requests. To generate the microbenchmark traces, we considered 4 load levels and 4 workload types. The load levels range from 1 to 4. The workload types are determined by the system call requested by the worker threads: i) random read (*pread* system call); ii) random write (*pwrite* system call); iii) sequential read (*read* system call); and iv) sequential write (*write* system call). Each request reads from or writes to files in blocks of 4096 bytes. Each worker thread performed a sequence of 5000 requests.

In the microbenchmark workload, to evaluate the interplay between the file system and the storage subsystem, we avoid having all the data set in memory. To this end, each worker thread operates over a different 10GB file, following the recommendation to use a file that is at least 4 times larger than the amount of available memory [11]; it reduces the probability that subsequent requests to random offsets will hit the same page cache.

To each combination of load level and workload type, we generated traces in three different modes: i) without pause between requests; ii) with a $10\mu sec$ pause between requests; and iii) with a $50\mu sec$ pause between requests.

The traces generated with the first mode are used by the compilation and the event-based replayer, with the *fullspeed* timing policy. The traces generated with the remaining modes are replayed by the event-based tool, with the *timestamp* timing policy.

The filebench workload we choose emulates the file system activity of a file-server in which 4 threads perform sequences of creates, deletes, appends, read, writes, and operations to file attributes (`stat` calls) to a directory tree. The target fileset under the directory tree is composed of 1000 files. The size of the files is given by a gamma distribution (with the mean and gamma parameters equal to 131072 and 1.5, respectively). Read and write operations operate over the whole file contents. The mean append size is of 16KB. The durations of each traced filebench execution was 30 seconds.

Incidentally, the filebench workload is more complex than the microbenchmark workload. For example, in the filebench workload, the size of the files vary, thus each thread performs a slightly different workload. Also, each file may be two orders of magnitude smaller than the files in the microbenchmark workload; thus, it is even possible that a file fits completely in the page cache at some point. Ultimately, this filebench workload highlights the impact of the benchmark working set in the file system performance.

In the following, we discuss the methods we applied to analyze the precision, bias and linearity of the event-based and compilation-based trace replay methods.

1) Determination of measurement precision: To account for measurement precision, we performed 10 replicated experiments to each generated trace. The result of the experiment is the average of the response time of all operations performed in the experiment.

We analyzed the precision of trace replay methods by comparing them to the precision of the reference values. In other words, we aimed to verify if the compilation-based and the event-based replayers increased variability when compared to the reference variability. To this end, we applied an F-test to compare replay precisions: for a measurement method A, let $u(R_w^A)$ be the precision of method A, that is, the uncertainty component due to random errors. In this way, we obtained evidence that method B has poorer precision than method A if $F_r > F_{\alpha,N_B-1,N_A-1}$, where F_r is the ratio between the variances of measurement methods:

$$F_r = \frac{u(R_w^B)^2}{u(R_w^A)^2}$$

and F_{α,N_B-1,N_A-1} is the critical value of the F distribution with $N_B - 1$ and $N_A - 1$ degrees of freedom at a significance level α [20].

2) Determination of measurement bias: The bias is the difference between the expected value of trace replay measurement and the reference values. To analyze the bias we compared the replay measurements and reference values we obtained in the determination of measurement precision, for the same combinations of workload level and workload type.

E. Calibration analysis

However useful the calibration may be, it is possible that, in some cases, it cannot be applied — trace replay included. This is because any calibration procedure is specific to the measurement instruments and to the environment used to determine the bias. Since the trace replay method is usually applied to evaluate a system different from the one used in the trace capture, or to evaluate changes in the environment, such as a new hardware component, the assumptions to apply the calibration do not hold.

Although in the trace replay case we cannot apply the results of the bias analysis to define a calibration procedure, these results can be used to identify problems on the design and implementation of the replay tools. By removing the sources of problems, it is possible to reduce the systematic errors.

F. Determination of measurement uncertainty

To estimate the uncertainty of the measurement method, the validation approach discriminates between the precision and bias (caused by random and systematic errors, respectively) found in the results of the experiments conducted as described in the Section III-D. The **within-laboratory reproducibility** component, $u(R_w)$, accounts for random errors, while the **bias uncertainty** component, $u(bias)$, accounts for systematic errors. These components are combined into a single metric – the **combined standard uncertainty**, u_c – given by the following equation [21]:

$$u_c = \sqrt{u(R_w)^2 + u(bias)^2}$$

Both components are calculated based on the execution of n replicated experiments. The experiment consists of replicated measurements of the measurand.

The within-laboratory reproducibility component, $u(R_w)$, is given by the standard deviation of the measurement results of the n experiments.

The bias uncertainty, $u(bias)$, is based on both the measurement results and on a collection of n reference values. These references are used to calculate the bias (the difference between a measured result and the reference value) in each experiment. According to this blueprint, the bias uncertainty $u(bias)$ is defined as

$$u(bias) = \sqrt{RMS_{bias}^2 + u(C_{ref})^2}$$

where RMS_{bias} accounts for the root mean square of the bias found in the n experiments and $u(C_{ref})$ stands for the uncertainty of the adopted reference values (although more reliable, there is still uncertainty in the reference measurements). For the batch of n experiments, the RMS_{bias} component is given by

$$RMS_{bias} = \sqrt{\frac{\sum_{i=1}^{n}(bias_i)^2}{n}}$$

For each experiment i, the $bias_i$ is given by the relative percentage error between the measured value x_i and the reference value ref_i: $bias_i = (x_i - ref_i)/ref_i \cdot 100$.

The $u(C_{ref})$ component is given by

$$u(C_{ref}) = \sqrt{\frac{\sum_{i=1}^{n} u(ref_i)^2}{n}}$$

where $u(ref_i)$ is the uncertainty of the reference value used in the i-th experiment.

The results of the uncertainty measurement are reported as $y_m \pm u_c$, where y_m is the mean value of the replicated experiments under controlled conditions. The combined standard uncertainty indicates that the measured results are within the interval $[y_m - u_c, y_m + u_c]$ with a confidence level of near 68%. For higher confidence levels, such as 95.5% and 99.7% there are **expanded uncertainties** of $y_m \pm 2 \cdot u_c$ and $y_m \pm 3 \cdot u_c$, respectively.

IV. Trace replayers uncertainty

In this section, we discuss the results of the validation of the trace replay methods, including the results of the characterization analysis as well as the determination of measurement uncertainty, for both the microbenchmark and the file-server workloads. To make the case of the metrology framework, we explain in more details the validation of the microbenchmark workload.

For this workload, in Section IV-A, we show the results of the characterization of trace replay in terms of precision (Section IV-A1), bias (Section IV-A2), and linearity of precision and bias (Section IV-A3). In Section IV-B, we focus on how to use the bias analysis to reduce systematic errors.

From Section IV-A to Section IV-B, we show in detail the results for the scenarios without pause between requests, for both the event and compilation-based replayers. The compilation-based replayer does not support the *timestamp* timing policy, thus, the scenarios with pause between requests are exclusive to the event-based replayer. The results for the latter are summarized in Section IV-C1 in addition to the overall measurement uncertainty of the scenarios without pause.

Finally, in Section IV-C2, we show the measurement uncertainty for the file-server workload, for both the event-based and compilation-based replayers.

A. Measurement Characterization

1) Measurement precision: In Table II we show the F_r ratio, as defined in Section III-D1, for compilation-based and event-based replayers, at the significance level of 0.05, and $N_B - 1$ and $N_A - 1$ degrees of freedom equal to 9 (based on 10 replicated experiments). The calculated value for F_{α,N_B-1,N_A-1} is 15.21.

As a rule, both the compilation and event-based replayers do not boost the sources of random errors. The compilation-based tool is less precise than the reference values only in 2 of 32 scenarios, while the event-based tool in less precise only in 3 of 32 scenarios. The sources of random errors, when present, affected the *sequential* instead of *random* workloads: this is because the magnitude of random errors is small in comparison with expected measured values for *random* workloads.

Table II: The compilation and event-based replayers are precise; according to an F-test at the significance level of 0.05, only in 5 of 64 scenarios the precision of trace replay measurements is lower than the precision of reference values (bold values are higher than an F_{α,N_B-1,N_A-1} of 15.21).

		Workload Level			
		1	2	3	4
Random read (RR)					
Compilation-based	`FS`	1.2	1.5	1.6	1.8
	`Temporal`	1.2	2.6	2.0	3.3
Event-based	`FS`	0.1	1.1	0.2	0.6
	`Temporal`	0.2	0.9	1.1	0.6
Random write (RW)					
Compilation-based	`FS`	1.4	6.0	3.8	4.1
	`Temporal`	1.4	9.1	4.7	4.0
Event-based	`FS`	0.3	0.8	0.4	2.9
	`Temporal`	2.4	0.9	0.3	3.2
Sequential read (SR)					
Compilation-based	`FS`	7.5	12.7	11.0	**18.6**
	`Temporal`	8.2	14.4	11.5	**20.7**
Event-based	`FS`	0.3	**17.3**	1.4	7.5
	`Temporal`	0.3	3.9	4.5	0.4
Sequential write (SW)					
Compilation-based	`FS`	0.9	0.7	0.5	0.6
	`Temporal`	0.9	1.3	0.3	0.7
Event-based	`FS`	0.2	**17.6**	13.3	10.8
	`Temporal`	0.4	5.8	**18.6**	9.1

2) Measurement bias: Figure 3 shows the average response time for the replicated trace replay experiments and the average reference values, for all combinations of workload levels and workload types.

A qualitative analysis of whether a given bias is acceptable or not is clearly specific to each measurement purpose. However, there is a purpose-independent criterion to decide if the bias is significant, and as consequence the measurement results should be subjected to bias correction. According to this criterion, the bias is significant if its magnitude is larger than twice the bias standard uncertainty [22].

Based on this criterion, the bias of compilation-based replayer is insignificant, for all combinations of workload type and levels. Based on the same criterion, the bias of the event-based replayer is significant is some scenarios. These scenarios include not only the expected *sequential read* workload type (which is very sensitive to systematic errors, since the measured values are small) but also *random write* scenarios.

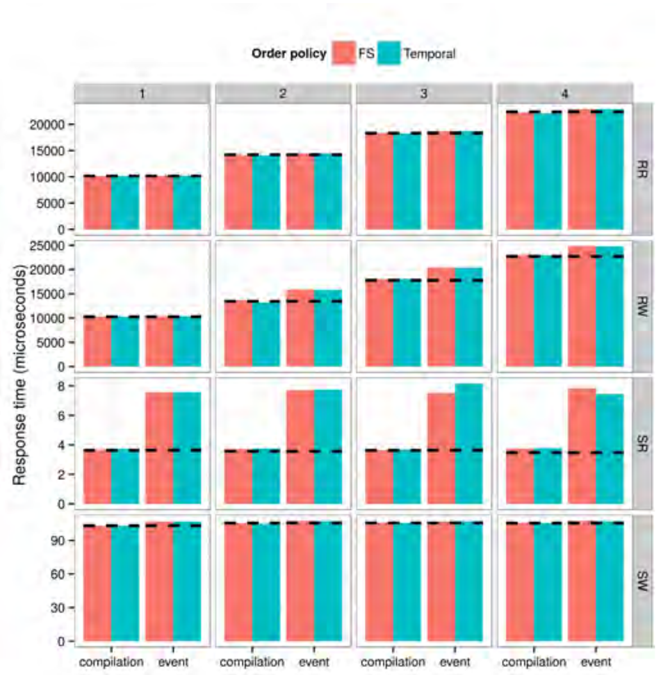

Figure 3: Bars show the average response time for trace replay measurements with the compilation-based and event-based tools. Dashed lines show average reference values. The bias is the difference between average response time and average reference values.

3) Linearity: In trace replay linearity we consider the impact of the increment of workload level on trace replay measurement errors. Table III and Table IV show this impact on the random and systematic errors, respectively. Table III reports the coefficient of variation of trace replay measurements, while Table IV reports the relative percentage between the average bias and the average reference value.

As a rule, there is no clear trend on the precision and on the bias due to the increment of the workload level. The exception is the replay of *random read* workloads by the event-based replayer. In this scenario, the relative bias strictly increased as the workload level increased. In any case, for the range of the workload level we considered, the magnitude of bias in this scenarios is not significant.

B. *Trace replay calibration (bias correction)*

As we described in Section III-E, although we cannot apply the results of the bias analysis to define a calibration procedure, these results can be used to identify problems on the design and implementation of the trace replayers that, when removed, may reduce the systematic errors. Based on the bias significance criterion adopted in Section IV-A2, the bias shown by event-based replays of *random write* and *sequential read* workloads indicate good targets for improvement.

The sources of systematic errors for *random write* and *sequential read* are likely to be different, as indicate the

Table III: Linearity of trace replay precision as the effect of workload level on the coefficient of variation of event-based and compilation-based replayes.

		Workload Level			
		1	2	3	4
Random read (RR)					
Compilation-based	FS	0.1%	0.2%	0.2%	0.4%
	Temporal	0.1%	0.2%	0.3%	0.5%
Event-based	FS	0.2%	0.7%	0.3%	0.4%
	Temporal	0.3%	0.6%	0.6%	0.4%
Random write (RW)					
Compilation-based	FS	0.1%	1.5%	0.4%	0.6%
	Temporal	0.1%	0.8%	0.4%	0.7%
Event-based	FS	0.2%	1.3%	0.5%	1.4%
	Temporal	0.5%	1.3%	0.5%	1.5%
Sequential read (SR)					
Compilation-based	FS	0.3%	0.4%	0.4%	0.2%
	Temporal	0.6%	0.2%	0.2%	0.3%
Event-based	FS	0.9%	9.6%	3.1%	6.7%
	Temporal	0.9%	4.5%	5.1%	1.6%
Sequential write (SW)					
Compilation-based	FS	0.1%	0.1%	0.1%	0.1%
	Temporal	0.1%	0.6%	0.1%	0.1%
Event-based	FS	0.2%	0.8%	0.4%	0.5%
	Temporal	0.2%	0.4%	0.4%	0.4%

Table IV: Linearity of trace replay bias as the relative percentage (to the average reference value) of event-based and compilation-based replayers.

		Workload Level			
		1	2	3	4
Random read (RR)					
Compilation-based	FS	0.1%	0.0%	0.3%	0.6%
	Temporal	0.1%	0.0%	0.3%	0.6%
Event-based	FS	1.1%	1.6%	2.2%	2.3%
	Temporal	1.1%	1.8%	2.1%	2.5%
Random write (RW)					
Compilation-based	FS	0.1%	0.0%	0.3%	0.6%
	Temporal	0.6%	1.8%	1.8%	1.2%
Event-based	FS	1.0%	17.9%	14.9%	9.2%
	Temporal	1.3%	17.4%	14.7%	9.0%
Sequential read (SR)					
Compilation-based	FS	1.8%	3.9%	0.9%	7.5%
	Temporal	2.7%	5.2%	2.0%	8.8%
Event-based	FS	108.1%	115.8%	107.2%	126.1%
	Temporal	109.0%	117.4%	125.1%	114.7%
Sequential write (SW)					
Compilation-based	FS	0.2%	0.1%	0.2%	0.2%
	Temporal	0.2%	0.5%	0.1%	0.0%
Event-based	FS	3.6%	1.7%	1.4%	2.0%
	Temporal	3.5%	1.5%	1.5%	1.5%

magnitude of bias shown in Figure 3; the bias for *sequential read* workload is no more than a few microseconds, while for the *random write* workload is at the millisecond scale.

Since the sources of errors are different, we must take different actions to mitigate them.

To reduce the systematic errors shown in the *sequential read* workload we evaluated the influence of the coordinator component adopted in the event-based replayer. We modified the original design by removing the coordinator component and distributing its responsibility among the worker threads. In the new design, the events that a worker thread will execute are assigned to it at the beginning of trace replay. A condition variable is associated to each event to enforce the ordering policy; this guard signalizes whether the predecessor of an event has already been executed.

To reduce the systematic errors shown in the *random write* workload we abandoned literature advice of running the replayer tool as a real time process [8]. As we described in Section III-B2, this was originally suggested as a way to prevent the replayer tool from being preempted by other tasks. This configuration generated at least one collateral effect in our experiments: the replayer worker threads exhibit a different scheduling pattern from the captured workload. After preemption (usually due to blocking for an IO operation), the replayer worker threads were taking longer to be resumed. This leads to larger response time as viewed by the applications.

Figure 4 shows the reduction of systematic errors, for the *random write* and *sequential read* workloads, due to the above modifications in the event-based trace replayer. Both modifications were effective. The first one, when applied to the *sequential read* workload, reduced the relative bias from up to 126.1% to no more than 15.6%. However useful for the *sequential read* workloads, this optimization was not able to deliver the same reduction of systematic errors for the *random write workloads*. This was because the typical measured value in the *random write* workload is one order of magnitude higher than the typical value in the *sequential read* workload; thus, the relative impact of the reduction of this error source in the *random write* workload is lower.

Only after abandoning the practice of running the trace replayer as a real time process we were able to achieve substantial reduction on the systematic errors for the *random write* workloads. Figure 4 shows that the relative bias is reduced from up to 17.9% to no more than 3.8%.

C. Determination of measurement uncertainty

In the previous sections we described the replay methods and characterized their error sources to replay a simple microbenchmark workload. In this section, after modifying the event-based replayer to reduce systematic errors, we complete the validation protocol by calculating the combined uncertainty u_c for the trace replay measurements of the microbenchmark (Section IV-C1) and a more complex, file-server workload (Section IV-C2).

1) Microbenchmark workload: Table V shows the combined uncertainty we found for the event-based and the compilation-based tools for all combinations of workload types and workload levels, as well as the ordering policies. This table reports the results for the scenarios without pause between requests, replayed with the *fullspeed* timing policy. For the event-based replayer, we also report the uncertainty before the changes we described in the previous section (in parentheses). The combined uncertainty, as described in Section III-F, is reported as $y_m \pm 2 \cdot u_c$, where y_m is the mean of the replicated experiments under controlled conditions. It is regarded as the expected uncertainty of measurements using the procedure we defined within our laboratory, at a 95.5% confidence level.

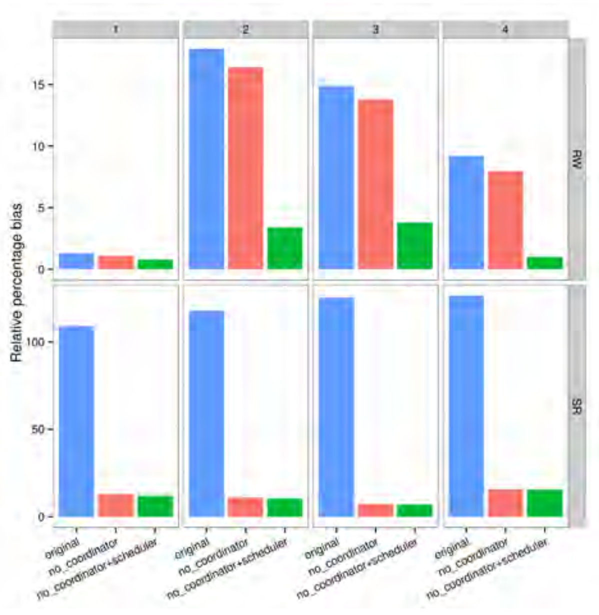

Figure 4: Relative percentage bias (to the average reference value) of the event-based replayer before (`original`) and after (`no_coordinator` and `no_coordinator + scheduler`) improvements. Bias is reported for *random write* and *sequential read* workloads. The workload levels range from 1 to 4.

Before the improvements made in the event-based replayer, the combined uncertainty of the compilation-based tool is lower than the combined uncertainty of the event-based tool. This rule holds true for all workload types, workload levels and replay policies. For the *random read* workload, the uncertainty of the compilation-based tool is up to 3.3%, and up to 5.1% to the event-based tool. For the *random write* workload, the uncertainty of the compilation-based tool is up to 9.1%, and up to 36.1% for the event-based tool. For the *sequential read* workload, the uncertainty of the compilation-based tool is no more than 20.7% while the uncertainty of the compilation-based replayer is up to 253.0%. For the *sequential write* workload, the uncertainty of the compilation-based replayer is up to 1.3%, and up to 7.2% for the event-based replayer.

After the improvements, the uncertainties of the compilation-based and the event-based tools were equivalent in most of the scenarios. Due to the reduction of systematic errors, the uncertainty of event-based replayer was greatly reduced. For example, with the original event-based design, the combined uncertainty of *sequential read* workload was up to 253%, while after the redesign, the uncertainty could be reduced to no more than 32.9%. Also, for the *random write* workload the original event-based uncertainty was up to 36.1%, while the uncertainty for the redesigned event-based replayer the uncertainty was no more than 10.9%.

In addition to the scenarios with no pause between the requests, we also show the uncertainty for the scenarios with pause between requests. Table VI reports the results for the event-based tool for all combinations of workload types and workloads levels, as well as the ordering policies replayed with the *timestamp* timing policy. We considered two different pause delays when generating the traces: i) $10\mu sec$; and ii) $50\mu sec$.

Table VI indicates that the uncertainty of the scenarios with pause between requests is higher than the uncertainty of the scenarios without pause. The reasons for this are threefold: i) the sources of errors we reported in Table V, which also act in the scenarios with pause, postpone the execution of the requests; ii) the common time delay functions used in trace replayers are imprecise (we used the `nanosleep` function); and iii) the *timestamp* timing policy defines that a request should be replayed as close as possible of the traced timestamps [9]. When combined, these conditions imply that the replayer, in many cases, does not wait before executing a request, since it is already delayed. As a consequence, replayer worker threads are less likely to be preempted by blocking, thus reducing response time; as Table VI shows, for the *random read* and *random write* workloads, the typical measured values are lower than the reference values (in contrast, Figure 3 shows that, for the scenarios without pause, the typical measured values are higher than then reference values).

2) Fileserver workload: After the analysis of the uncertainty of the microbenchmark workload and the improvements made to the event-based replayer, we analyzed the uncertainty to replay the file-server workload. Table VII shows the combined uncertainty we found for the event-based and the compilation-based tools. In addition to the combined uncertainty, we also show the mean reference value as observed in the file-server traces. As in the analysis of microbenchmark traces, we report the combined uncertainty as $y_m \pm 2 \cdot u_c$, where y_m is the mean of the replicated experiments under controlled conditions, at a 95.5% confidence level.

Table VII shows only the results for the *FS* ordering policy; as in the previous analysis, there is no significant difference between the results of *FS* and *Temporal* ordering policies. Also, Table VII shows only the uncertainty for the *read* and *write* operations which have more impact on the performance of the file-server workload than the metadata operations such as file creation and file removal.

Table VII indicates that, the accuracy of the replay of the file-server workload, for both the event-based and the compilation-based tools, is lower than the accuracy of the replay of the simpler microbenchmark workload.

For example, for the *read* operations of the file-server workload, the uncertainty of the compilation-based tool

Table VI: Combined measurement uncertainty $y_m \pm 2 \cdot u_c$ at a 95.5% confidence level for the event-based replay of the microbenchmark workload with pause between requests.

	Event-based		
Workload Level	`FS`	`Temporal`	Reference
	Thread request delay = $10\mu s$		
	Random read (RR)		
1	$10435.8 \pm 1.4\%$	$10435.1 \pm 1.5\%$	10414.95
2	$14744.2 \pm 2.3\%$	$14720.6 \pm 2.4\%$	14823.34
3	$18543.7 \pm 4.5\%$	$18551.4 \pm 4.4\%$	18934.93
4	$22727.3 \pm 4.2\%$	$22750.4 \pm 4.0\%$	23169.52
	Random write (RW)		
1	$10579.3 \pm 5.2\%$	$10683.6 \pm 7.8\%$	10555.51
2	$13955.5 \pm 10.4\%$	$13513.4 \pm 14.8\%$	14458.26
3	$18375.7 \pm 14.1\%$	$18410.0 \pm 15.6\%$	19425.76
4	$23529.7 \pm 11.1\%$	$23071.2 \pm 14.2\%$	24582.37
	Sequential read (SR)		
1	$4.0 \pm 10.4\%$	$4.0 \pm 10.4\%$	3.84
2	$3.8 \pm 18.7\%$	$3.8 \pm 19.0\%$	3.66
3	$3.4 \pm 19.2\%$	$3.4 \pm 18.3\%$	3.61
4	$3.8 \pm 20.6\%$	$3.8 \pm 20.8\%$	3.57
	Sequential write (SW)		
1	$100.7 \pm 2.3\%$	$100.7 \pm 2.2\%$	101.78
2	$103.5 \pm 1.8\%$	$103.4 \pm 2.3\%$	104.43
3	$103.5 \pm 1.7\%$	$103.5 \pm 1.6\%$	104.28
4	$103.7 \pm 1.3\%$	$103.8 \pm 1.1\%$	104.30
	Thread request delay = $50\mu s$		
	Random read (RR)		
1	$10394.6 \pm 1.3\%$	$10399.4 \pm 1.4\%$	10372.18
2	$14625.0 \pm 4.9\%$	$14651.1 \pm 4.6\%$	14925.17
3	$18741.7 \pm 4.8\%$	$18706.0 \pm 5.1\%$	19150.70
4	$22804.2 \pm 7.2\%$	$22746.7 \pm 7.7\%$	23624.56
	Random write (RW)		
1	$10610.6 \pm 10.5\%$	$10611.8 \pm 10.0\%$	10881.93
2	$14009.2 \pm 9.9\%$	$13748.1 \pm 12.8\%$	14510.74
3	$18304.0 \pm 16.9\%$	$17670.9 \pm 22.5\%$	19723.34
4	$23070.0 \pm 21.5\%$	$23056.5 \pm 21.8\%$	25677.56
	Sequential read (SR)		
1	$4.1 \pm 28.3\%$	$4.1 \pm 28.3\%$	3.67
2	$3.8 \pm 7.9\%$	$3.9 \pm 7.9\%$	3.80
3	$3.8 \pm 13.7\%$	$3.9 \pm 15.3\%$	3.76
4	$4.0 \pm 25.9\%$	$4.0 \pm 27.5\%$	3.60
	Sequential write (SW)		
1	$101.1 \pm 2.0\%$	$101.2 \pm 1.7\%$	102.00
2	$103.6 \pm 1.8\%$	$103.1 \pm 2.6\%$	104.48
3	$103.7 \pm 1.2\%$	$103.5 \pm 1.5\%$	104.20
4	$103.8 \pm 1.1\%$	$103.6 \pm 1.6\%$	104.37

Table V: Combined measurement uncertainty $y_m \pm 2 \cdot u_c$ at a 95.5% confidence level for the replay of the microbenchmark workload. Measured values are shown in microseconds. In parentheses we show combined uncertainty for the event-based tool before improvements to reduce systematic errors.

	Event-based		Compilation-based	
Workload Level	`FS`	`Temporal`	`FS`	`Temporal`
	Random read (RR)			
1	$10221.2 \pm 2.0\%$ $(10253.7 \pm 2.5\%)$	$10214.2 \pm 1.8\%$ $(10257.2 \pm 2.6\%)$	$10154.1 \pm 1.2\%$	$10155.2 \pm 1.2\%$
2	$14255.1 \pm 1.9\%$ $(14437.4 \pm 3.8\%)$	$14252.4 \pm 1.7\%$ $(14466.2 \pm 4.1\%)$	$14201.2 \pm 1.5\%$	$14100.2 \pm 2.6\%$
3	$18477.8 \pm 2.2\%$ $(18727.6 \pm 4.6\%)$	$18507.9 \pm 2.9\%$ $(18710.0 \pm 4.6\%)$	$18371.8 \pm 1.6\%$	$18235.2 \pm 2.0\%$
4	$22579.0 \pm 2.4\%$ $(22891.6 \pm 4.8\%)$	$22627.1 \pm 2.7\%$ $(22925.6 \pm 5.1\%)$	$22243.5 \pm 1.8\%$	$22046.4 \pm 3.3\%$
	Random write (RW)			
1	$10286.5 \pm 0.9\%$ $(10357.9 \pm 2.2\%)$	$10335.7 \pm 3.4\%$ $(10389.0 \pm 2.9\%)$	$10293.4 \pm 1.4\%$	$10312.3 \pm 1.4\%$
2	$13909.0 \pm 10.9\%$ $(15871.6 \pm 36.1\%)$	$13846.5 \pm 8.6\%$ $(15803.5 \pm 35.1\%)$	$13660.0 \pm 6.0\%$	$13211.2 \pm 9.1\%$
3	$18435.0 \pm 9.4\%$ $(20411.4 \pm 29.8\%)$	$18117.6 \pm 5.4\%$ $(20372.9 \pm 29.4\%)$	$18024.8 \pm 3.8\%$	$18089.9 \pm 4.7\%$
4	$22946.1 \pm 3.2\%$ $(24807.6 \pm 18.7\%)$	$22915.4 \pm 4.3\%$ $(24764.2 \pm 18.4\%)$	$23076.0 \pm 4.1\%$	$22988.3 \pm 4.0\%$
	Sequential read (SR)			
1	$4.1 \pm 24.9\%$ $(7.6 \pm 216.4\%)$	$4.1 \pm 24.7\%$ $(7.6 \pm 218.1\%)$	$3.7 \pm 7.5\%$	$3.7 \pm 8.2\%$
2	$3.9 \pm 23.2\%$ $(7.7 \pm 233.0\%)$	$3.9 \pm 22.8\%$ $(7.8 \pm 235.4\%)$	$3.7 \pm 12.7\%$	$3.8 \pm 14.4\%$
3	$3.9 \pm 17.5\%$ $(7.5 \pm 214.7\%)$	$3.9 \pm 17.4\%$ $(8.2 \pm 250.8\%)$	$3.7 \pm 11.0\%$	$3.7 \pm 11.5\%$
4	$4.0 \pm 32.9\%$ $(7.8 \pm 253.0\%)$	$4.0 \pm 32.4\%$ $(7.4 \pm 229.7\%)$	$3.7 \pm 18.6\%$	$3.8 \pm 20.7\%$
	Sequential write (SW)			
1	$102.5 \pm 1.6\%$ $(106.9 \pm 7.2\%)$	$102.6 \pm 1.4\%$ $(106.9 \pm 7.1\%)$	$103.0 \pm 0.9\%$	$103.0 \pm 0.9\%$
2	$105.4 \pm 0.6\%$ $(107.5 \pm 3.8\%)$	$105.2 \pm 1.6\%$ $(107.2 \pm 3.2\%)$	$105.7 \pm 0.7\%$	$105.1 \pm 1.3\%$
3	$105.3 \pm 1.2\%$ $(107.1 \pm 2.8\%)$	$105.5 \pm 0.6\%$ $(107.3 \pm 3.1\%)$	$105.9 \pm 0.5\%$	$105.6 \pm 0.3\%$
4	$105.6 \pm 1.3\%$ $(107.7 \pm 4.2\%)$	$105.8 \pm 0.6\%$ $(107.1 \pm 3.2\%)$	$105.8 \pm 0.6\%$	$105.5 \pm 0.7\%$

is up to 94.79% while the uncertainty of the same tool for the *read* workload in the microbenchmark (**SR**) is no more than 20.7%. For the *write* operations of the file-server workload, the uncertainty of the compilation-based tool is up to 33.79% while the uncertainty for the write workload in the microbenchmark (**SW**) is no more than 1.3%. Note that, as we described in Section III-D, *read* and *write* operations in the file-server workload are sequential ones.

For the event-based tool, the uncertainty to replay the *read* operations of the file-server workload is up to 120.97% while the uncertainty to replay the read workload (**SR**) in the microbenchmark is no more 32.9%. For the event-based tool, the uncertainty to replay the *write* operations of the file-server workload is up to 124.03% while the uncertainty to replay the read workload (**SW**) in the microbenchmark is no more 1.4%.

Table VII: Combined measurement uncertainty $y_m \pm 2 \cdot u_c$ at a 95.5% confidence level for the replay of the file-server benchmark. Trace replay tools applied the *FS* ordering policy in combination with the *fullspeed* timing policy.

	Event-based	Compilation-based	Reference
Read	$20.73 \pm 118.27\%$	$27.21 \pm 92.72\%$	50.72
Write	$50.45 \pm 79.81\%$	$69.79 \pm 33.79\%$	83.95

We noticed that two sources of errors account for the uncertainty of the file-server workload. The first source is negative (it reduces the measurand) and greatly contributes to the bias. The second source is positive (it increases the measurand) and explains the difference between the uncertainty of the event and the compilation-based replays.

The first source of error is caused by a difference on the capture and replay environments. More specifically, the memory footprints of the replay tools are smaller than the memory footprint of the filebench benchmark. As a result, during the trace replay, there is more memory available to the page cache and the file system operations are more likely to hit the memory subsystem instead of the disk, thus reducing the file system response time.

The second source of error is related to the limitations of the replay tools to correctly match the concurrency of the traced workload. Figure 5 shows the empirical cumulative function of the concurrency level, as the number of active requests at a given time, that we found for the captured workload and the replayed workloads for both the event-based and the compilation-based replayers.

Figure 5 shows that the event-based replayer reduces the concurrency while the compilation-based one increases it, in comparison with the concurrency of the reference workload. There is no more than one active request in up to 75% of the requests replayed with the event-based tool, while for the compilation-based tool in more than half of time there are at least 3 active requests. The higher the concurrency, the higher is the chance to have a thread being preempted by another worker thread, thus increasing its response time by the off-cpu period.

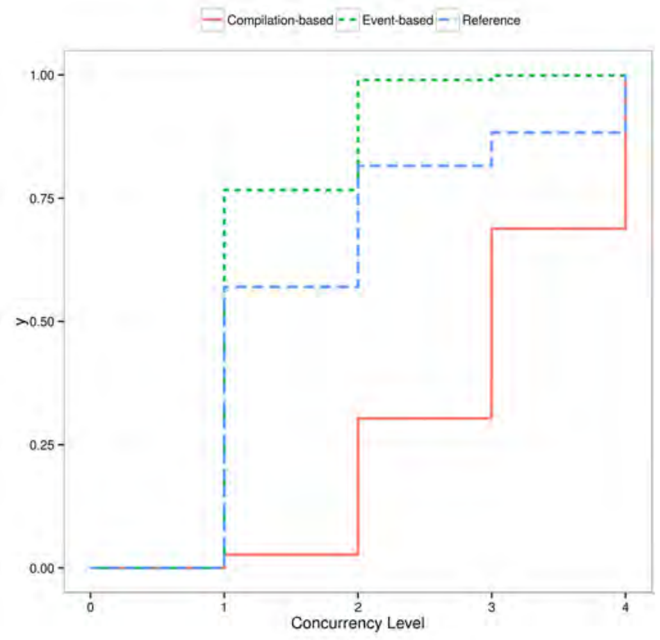

Figure 5: Empirical cumulative distribution function of the concurrency level for the replays of the file-server workload and the reference workload. The number of current requests is given by number of active requests at a given time.

V. Discussion

The state-of-practice in file system trace replay is marked by a number of disparate replay tools, designs and measurements procedures. In this scenario, by characterizing the quality of the available methods, the validation protocol not only helps guiding the choice of a trace replay alternative but also helps refining replay best practices — ultimately, this contributes towards the development of a standard trace replay method.

An important finding of this case study was that some known measurement procedures are flawed. In particular, we observed that the choice of operating system scheduler policy, prescribed as good practice in the literature to isolate trace replayer processes from the influence of other processes in the experimental environment (in metrology terms, to increase the **selectivity** of the trace replay method), has a collateral effect on the systematic errors.

We argue that we can abandon this procedure nowadays. In most cases, the experimental environment where the traces are replayed is well controlled. That is, it is possible to avoid running disturbing applications and services. More important, this finding also indicates that we must be careful with practices that used to work in the past. This is because, operating system and hardware also impact replay methods, and these past conditions may be different today.

After adopting the best practices and optimizing the implementation of the trace replayers to reduce systematic errors, as described in Section IV-B, the event and compilation based replayers are roughly equivalent to replay the microbenchmark workload, based on the uncertainty shown in Table V.

For the *random read* and *sequential write* workloads, there is no clear difference between the uncertainty of the replayers. Also, the uncertainty is low in both cases.

If one targets *sequential read* workloads, we also argue that the compilation and event-based replayers are equivalent choices (however, due to a more subtle reason). In this scenario, even after the improvements made in the event-based replayer, the uncertainty of the compilation-based tool is clearly lower, for all combinations of workload level and replayer policies. However, the difference between the expected measured values is within the microsecond range. That is, although we have a considerable difference in the relative bias, the difference we observe in the bias magnitude is unlikely to lead to any practical consequence for the performance analysis of file systems.

When targeting *random write* workloads, one faces a situation similar to the *random read* case, where the event-based replayer shows similar uncertainty to the compilation-based replayer uncertainty. However, this uncertainty, in some scenarios, is higher than *random read* uncertainty: it is up to 10.9% for the event-based replayer and it is up to 9.1% for the compilation-based replayer. For some specific uses, it may could be the case that neither of the tools is appropriate to be used in this scenario.

We found that the event-based trace replayer has limitations to replay traces with pause between requests — the uncertainty is higher than to replay traces without pause. For example, for *random read* workload without pause the combined uncertainty is up to 5.1%, while for *random read* workload with pause it is up to 7.7%. For the *random write* workload without pause the uncertainty is no more than 10.9, while for the corresponding workload with pause it is up to 22.5%. This finding indicates that, when one measures a sensitive metric such as the one we adopted in this case study (response time), the common linux keeping functions may be inappropriate to the *timestamp* timing policy. In metrology terms, this imposes a **limit of quantification** at sub-millisecond range. One possible workaround to reduce the uncertainty in these scenarios is to adopt a timing control with higher resolution and lower overhead, although less portable, such as busy-wait loops based on TSC hardware counter.

Another important finding is that, similarly to our observations for trace capture [13], in some cases, to accurately replay file system traces it is important to collect more information from the traced environment, in addition to the file system activity. As the uncertainty of the file-server workload shows, resource usage such as the amount of allocated memory may affect the measurement results.

In addition to improving the quality of the trace replay methods, the validation results can also be used in other practical situations. For example, an important application is to find atypical measurement results. Naturally, in these cases, the causes of the aberrant results should be investigated. However, it is also desirable to report an acceptable result despite the atypical measurements. There are metrology protocols that help calculating the additional number of experiments to find such acceptable result and to state the obtained results [20].

Another practical application of metrology is to check the stability of measurement results [20]. This is specially important when the measurements are assessed over a long period. In this scenario, trace replay measurements are periodically executed, for example, to evaluate changes in the design of the file system or the impact of recently collected traces. During this long run, the operator may fail to correctly follow the measurement procedure, for example, when configuring the file system or preparing the traced data. In addition to operation errors, unnoticed updates in the system software and hardware malfunctioning may happen during the long run. All these factors can increase the random systematic errors beyond the expected limits defined in the method validation study.

VI. Related work

In this section, we review the related literature in trace-based file system performance evaluation. We are interested here on how past work relate to metrology concepts (even if not explicitly declared), rather than discussing their effectiveness. We also review computer science literature – not related to file system performance evaluation – that has explicitly applied a branch of metrology in their research methodology.

`DFSTrace` [23] is a kernel-based trace capture tool developed under the umbrella of the Coda file system project [24]. The `DFSTrace` authors referred to trace capture bias as *overhead*. They determined the overhead by collecting the elapsed time to run the Andrew benchmark over a file system without instrumentation, and comparing this to the time required to run the same workload over an instrumented file system. Their evaluation is in fact a bias estimation. Nevertheless, they failed to follow metrology advice concerning correction of the estimated bias.

Trace-based tools are popular not only in the file system area. For example, Müller et al. developed the `Vampir` tool set to analyze MPI parallel programs [25]. Similarly to `DFSTrace` authors, they evaluated `Vampir` trace capture bias as overhead. They determined the bias by comparing the elapsed time to run MPI programs without instrumentation with the elapsed time to run them after instrumentation. They observed considerable bias in most of the evaluated cases. As the `DFSTrace` authors, Müller et al. also missed the correction of capture measurements by calibration.

Later, in a joint performance measurement effort by the HPC community, the `Vampir` trace capture tool was replaced in favor of the `Score-P` instrumentation tool [26]. The main improvement of the latter alternative was to store data in pre-allocated memory chunks. This improvement minimized the overhead of allocating these chunks at runtime, as observed by comparing the runtime of running a MPI program without instrumentation with the runtime to run the same program with `Vampir` and `Score-P` instrumentation.

Anderson et al. developed `Buttress` to improve timing control in IO trace replays [27]. They analyzed replay timing by measuring the issue error, i.e. the difference between the intended timestamp to replay a request and the actual replayed timestamp. They also evaluated issue error as a function of the number of I/O replayed requests per seconds. This is in the direction of metrology **linearity** analysis.

The `TBBT` event-based replayer was the first tool that has dealt with the differences between capture and replay environments [9]; in metrology terms, **reproducibility** concerns. To cope with this requirement, they developed ordering and timing policies to shape replayed workloads. Although `TBBT` policies are a solid proposal, the authors overlooked evaluating the fidelity of their system. Instead, they evaluated `TBBT` ability to scale request dispatching (however, more recently, Pereira et al. have shown that the choice of `TBBT` timing and ordering policies have a big impact on measured performance [10]). As the work by Anderson [28], scalability analysis is related to metrology definitions of **rated** and **limiting operating conditions**.

Zev et al. revisited the order discovery problem with the `ARTC` replayer [7]. They suggested collecting extra information, such as process and thread ids, to infer request order. Similar to our approach, they evaluated their `ARTC` replayer by comparing the time to execute microbenchmark programs with the time to replay the corresponding trace. They also evaluated `ARTC` behavior when trace replay takes place in an environment different from the one used in the capture — in metrology terms, they evaluated the measurement method under **reproducible conditions**.

Metrology definitions and methods are not tied to any particular application. As a result, metrology has been broadly applied in physics [29], [30], chemistry [31], [32] and biology [33], for example. Although not as mainstream as it became in other sciences, metrology has already been explicitly applied in computer science area (in particular, using the GUM modeling approach instead of the empirical approach we adopted in our case study). Betta et al. evaluated the uncertainty in face recognition systems [34]. Metrology was also applied to the dependability analysis of distributed systems. For example, the synchronization uncertainty analysis of reliable clocks [35].

VII. Conclusions

This paper reported a validation of trace replay methods through a metrology protocol popular in other sciences. The validation indicated that both the event and compilation-based methods we evaluated provide good precision however the event-based method shows significant bias. The results of the validation also guided the reduction of the bias by: i) identifying limitations in the concurrent access to replayer data structures, and ii) showing that the practice of controlling the operating system scheduler policy to isolate the replay process has a collateral effect on measurement bias.

We considered workloads generated by a micro and macrobenchmarks to validate the trace replay methods. Although these workloads allowed us to find problems in current practice and to solve some trace replay shortcomings, we are not able to tell the uncertainty to replay more complex traces. We plan to apply the same metrology framework to evaluate other workloads, including replays of real applications.

In the metrology validation protocol we adopted, the variability of measurement results, calculated in the precision component of the measurement uncertainty, arises despite the tight controlled experimental conditions — the protocol assumes repeated experiments in which identical materials are applied with no changes in the environment nor in the measurement instruments. However, when used in practice, some factors can increase the variability of measurement results even more. This is the case, for example, of applying different machines to replay the traces — very common when one needs to compare results obtained by different laboratories. Since the protocol we adopted cannot estimate these uncertainty sources, the accounted uncertainty is, in fact, a lower bound. We plan to extend our case study by considering uncertainty arising from these other uncertainty sources, leading to a **reproducibility** metrology analysis.

References

[1] E. Shriver, A. Merchant, and J. Wilkes, "An analytic behavior model for disk drives with readahead caches and request reordering," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 1, pp. 182–191, 1998.

[2] E. K. Lee and R. H. Katz, "An Analytic Performance Model of Disk Arrays," *SIGMETRICS Perform. Eval. Rev.*, vol. 21, no. 1, pp. 98–109, 1993.

[3] S. Frølund and P. Garg, "Design-time Simulation of a Large-scale, Distributed Object System," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 4, pp. 374–400, 1998.

[4] C. A. Thekkath, J. Wilkes, and E. D. Lazowska, "Techniques for File System Simulation," *Softw. Pract. Exper.*, vol. 24, pp. 981–999, 1994.

[5] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-structured File System," *ACM Trans. Comput. Syst.*, vol. 10, pp. 26–52, 1992.

[6] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open Versus Closed: A Cautionary Tale," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation*, NSDI'06, pp. 18–18, 2006.

[7] Z. Weiss, T. Harter, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "ROOT: Replaying Multithreaded Traces with Resource-oriented Ordering," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, SOSP'13, pp. 373–387, 2013.

[8] N. Joukov, T. Wong, and E. Zadok, "Accurate and Efficient Replaying of File System Traces," in *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies*, FAST'05, pp. 337–350, 2005.

[9] N. Zhu, J. Chen, and T.-C. Chiueh, "TBBT: Scalable and Accurate Trace Replay for File Server Evaluation," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, FAST'05, pp. 323–336, 2005.

[10] T. E. Pereira, L. Sampaio, and F. V. Brasileiro, "On the Accuracy of Trace Replay Methods for File System Evaluation," in *Proceedings of the IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pp. 380–383, 2013.

[11] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A Nine Year Study of File System and Storage Benchmarking," *Trans. Storage*, vol. 4, pp. 5:1–5:56, 2008.

[12] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer, "Benchmarking File System Benchmarking: It *IS* Rocket Science," in *13th USENIX Conference on Hot Topics in Operating Systems*, pp. 9–9, 2011.

[13] T. E. Pereira, F. Brasileiro, and L. Sampaio, "A study on the errors and uncertainties of file system trace capture methods (to apear)," in *Proceedings of the 9th ACM International Systems and Storage Conference*, 2016.

[14] M. Tarihi, H. Asadi, and H. Sarbazi-Azad, "DiskAccel: Accelerating Disk-Based Experiments by Representative Sampling," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.*

[15] "Filebench." http://filebench.sourceforge.net/wiki/index.php/Main_Page. Accessed: 2016-02-12.

[16] B. I. des Poids et Mesures, C. électrotechnique internationale, and O. internationale de normalisation, *Guide to the Expression of Uncertainty in Measurement.* International Organization for Standardization, 1995.

[17] EURACHEM, Teddington (UK), *Eurachem Guide: The Fitness for Purpose of Analytical ¡ethods – A Laboratory Guide to Method Validation and Related topics*, 1998.

[18] Joint Committee for Guides in Metrology (JCGM), *International vocabulary of metrology. Basic and general concepts and associated terms (VIM), 200:2012*, 3rd ed., 2012.

[19] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel.* Sebastopol, CA: O'Reilly Media, Inc., 2nd ed., 2002.

[20] International Organization for Standardization, *ISO 5725-6: Accuracy (Trueness and Precision) of Measurement Methods and Results - Part 6 : Use in practice of accuracy values*, 1994.

[21] Nordic Innovation Centre, Taastrup, Denmark, *Handbook for Calculation of Measurement Uncertainty in Environmental Laboratories*, 2012.

[22] W. Hasselbarth, *Guide to the Evaluation of Measurement Uncertainty for Quantitative Test Results.* Paris: European Federation of National Associations of Measurement, Testing and Analytical Laboratories, 2006.

[23] L. Mummert and M. Satyanarayanan, "Long term distributed file reference tracing: Implementation and experience," tech. report dtic document, Carnegie Mellon University, 1994.

[24] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, vol. 39, pp. 447–459, 1990.

[25] M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel, "Developing Scalable Applications with Vampir, VampirServer and VampirTrace.," in *Advances in Parallel Computing*, pp. 637–644, 2007.

[26] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. D. Malony, *et al.*, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir.," in *Tools for High Performance Computing*, pp. 79–91, Berlin: Springer, 2011.

[27] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan, "Buttress: A toolkit for flexible and high fidelity I/O benchmarking," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, FAST'04, pp. 45–58, 2004.

[28] E. Anderson, “Capture, conversion, and analysis of an intense NFS workload,” in *Proccedings of the 7th USENIX Conference on File and Storage Technologies*, FAST’09, pp. 139–152, 2009.

[29] S. Rabinovich, “General information about measurements,” in *Measurement Errors and Uncertainties: Theory and Practice*, pp. 1–28, New York: Springer-Verlag, 3rd ed., 2005.

[30] P. Fornasini, *The Uncertainty in Physical Measurements: An Introduction to Data Analysis in the Physics Laboratory.* New York: Springer-Verlag, 1st ed., 2008.

[31] D. Massart, B. Vandeginste, L. Buydens, S. D. Jong, P. J. Lewi, and J. Smeyers-Verbeke., “Handbook of Chemometrics and Qualimetrics: Part A,” in *Data Handling in Science and Technology*, vol. 20, pp. 1–867, New York: Elsevier, 1997.

[32] H. Czichos, T. Saito, and L. E. Smith, *Springer Handbook of Metrology and Testing.* Berlin: Springer-Verlag, 2nd ed., 2011.

[33] Office for Official Publications of the European Communities, Luxembourg, *Metrology in Chemistry and Biology: A Practical Approach*, 1998.

[34] G. Betta, D. Capriglione, M. Corvino, C. Liguori, and A. Paolillo, “Estimation of influence quantities in face recognition,” in *IEEE International Instrumentation and Measurement Technology Conference*, pp. 963–968, 2012.

[35] A. Bondavalli, A. Ceccarelli, and L. Falai, “Assuring Resilient Time Synchronization,” in *Proceedings of the 2008 Symposium on Reliable Distributed Systems*, pp. 3–12, 2008.