DOI: 10.14529/jsfi180102

Record and Replay Techniques for HPC Systems: A Survey

Dylan Chapp[1], *Kento Sato*[2], *Dong H. Ahn*[2], *Michela Taufer*[1]

© The Authors 2018. This paper is published with open access at SuperFri.org

Record-and-replay techniques provide the ability to record executions of nondeterministic applications and re-execute them identically. These techniques find use in the contexts of debugging, reproducibility, and fault-tolerance, especially in the presence of nondeterministic factors such as message races. Record-and-replay techniques are highly diverse in terms of the fidelity of replay they provide, the assumptions they make about the recorded application, the programming models they target, and the runtime overheads they impose. In the high performance computing (HPC) environment, all the above factors must be considered in concert, thus presenting additional implementation challenges. In this manuscript, we survey record-and-replay techniques in terms of the programming models they target and the workloads on which they were evaluated, providing a categorization of these techniques benefiting application developers and researchers targeting exascale challenges. This manuscript answers three questions through this survey: What are the gaps in the existing space of record-and-replay techniques? What is the roadmap to widespread use of record-and-replay on production-scale HPC workloads? And, what are the critical open problems that must be addressed to make record-and-replay viable at exascale?

Keywords: reproducibility, nondeterminism, fault-tolerance, exascale, message-passing, shared memory, proxy application, HPC benchmarks.

Introduction

Record-and-replay (R&R) techniques provide the ability to monitor and record changes in program state over one execution (i.e., the recorded execution) of an application and reproduce those changes–and thus, the behavior of the application–during a subsequent execution (i.e., the replayed execution). The convergence of extremely high levels of hardware concurrency, the effective overlap of computation and communication, and overall code complexity make R&R a vital tool for coping with non-determinism in HPC applications. Non-determinism in HPC applications is a growing problem [1, 15, 48] and it manifests at multiple levels. Non-determinism may manifest in low-level communication primitives (e.g., the inherent non-determinism of non-blocking matching functions in MPI); it may manifest in libraries (e.g., dynamic load-balancing libraries [29]); or it may manifest at the application level (e.g., Monte-Carlo simulations). The common motivation of all R&R techniques is to manage these forms of non-determinism. Specifically, R&R techniques play key roles in three different HPC contexts. First, these techniques are used for debugging parallel programs that exhibit non-deterministic bugs during a code execution when moving, for example, from a smaller to a larger scale or from one platform to a different one. Second, R&R is useful during testing parallel programs, where users may want to ensure numerical or scientific repetition of certain results despite, for example, not enforcing specific message interleavings. Third, R&R techniques find alternative use in the context of fault-tolerance where the execution of one or more processes may need to be rolled back and replayed to recover from a fault.

This survey presents a rigorous classification of R&R techniques based on the programming model the R&R technique targets (i.e., shared memory or distributed memory) and the workloads on which the techniques have been evaluated (i.e., theoretical, non-HPC, HPC benchmarks, and HPC applications). For each programming model, we refine our categorization based

[1]University of Delaware, Newark, United States
[2]Lawrence Livermore National Laboratory, Livermore, United States

on the overall approach the technique takes (i.e., data-replay, order-replay, or a hybrid of the two). This dimension is shown in Fig. 1 and 2 on the horizontal axis. The vertical axis roughly corresponds to time, as we show the evolution of R&R techniques through conceptual eras. For each technique, we indicate the category of workloads it is evaluated against. We consider four categories: first, an absence of empirical evaluation (i.e., for purely theoretical works); second, non-HPC workloads (e.g., network servers, as these are common evaluation workloads for shared memory techniques); third, HPC benchmarks (e.g., the NAS Parallel Benchmarks); and fourth, full HPC applications. In Fig. 1 and 2 we represent the paper that introduced each technique as a box. The evaluation workloads are indicated by symbols in the boxes. For techniques that are evaluated against multiple categories of workloads, multiple symbols are shown. Additionally, due to the impact of logical clock algorithms on R&R (both for shared and distributed memory techniques) and interest in leveraging specialized hardware (for shared memory techniques only), we color the boxes corresponding to use of these features.

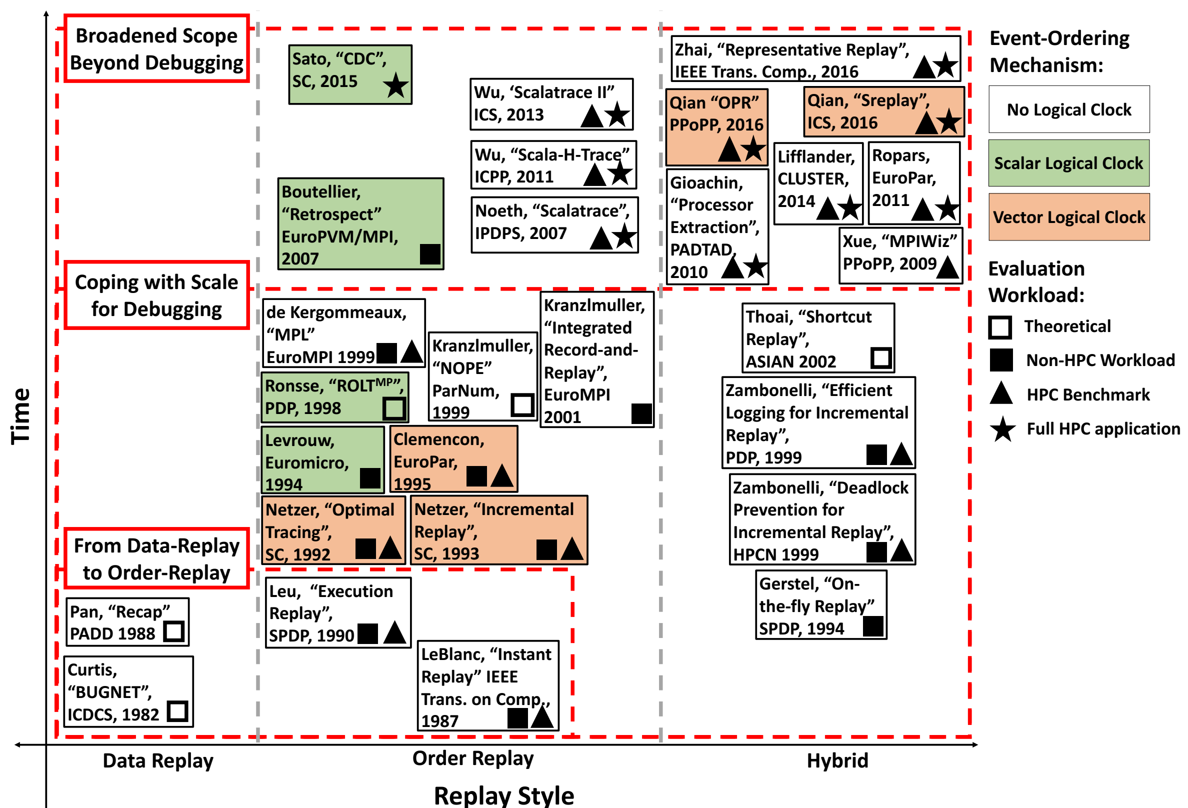

Figure 1. Record-and-replay techniques for distributed memory programming models

Our survey targets three communities: 1) researchers who are interested in filling the gaps in the existing space of R&R techniques; 2) developers and maintainers of HPC applications who use R&R primarily as a debugging aid; 3) the community of researchers exploring uses of R&R at exascale, including for fault-tolerance, resilience, and reproducibility. For each of these communities we identify a guiding question that this survey answers. Our survey supports researchers attempting to design R&R techniques that address so-far-unexplored regions of the technique space. They will benefit from the taxonomy that this survey develops by learning how to situate their work. The guiding question for this community is: Where are the gaps in the technique space? An HPC developer benefits from this survey because it informs them about which R&R techniques apply to their workloads. The guiding question for this community is:

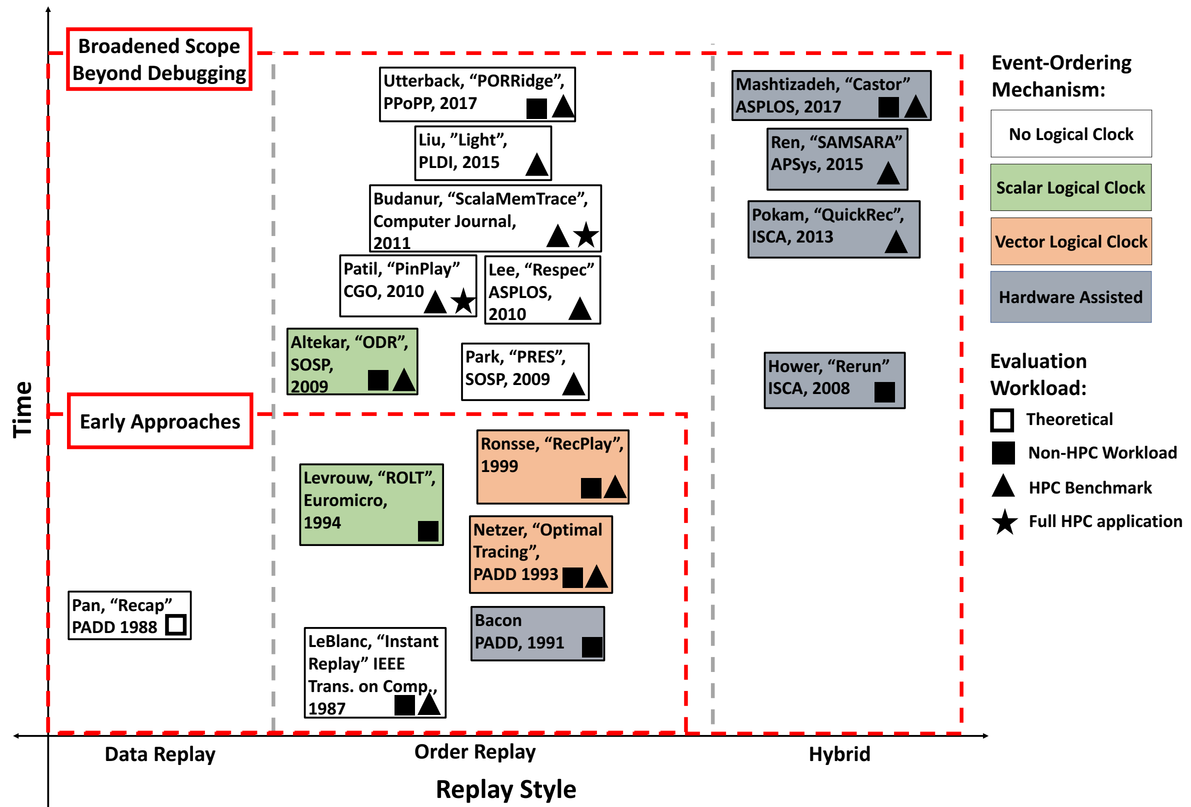

Figure 2. Record-and-replay techniques for shared memory programming models

What are the workloads to which R&R techniques apply? Finally, an exascale-centric researcher benefits from this survey by gaining a holistic view of how R&R techniques have overcome scaling challenges. The guiding question for this community is: What are the open problems standing between the state-of-the-art in R&R and suitability for future exascale systems?

The remainder of the article is structured in the following way. In Section 1, we provide terminology and definitions needed for evaluating R&R techniques. In Section 2, we discuss techniques targeting shared memory programming models. In Section 3, we discuss R&R techniques targeting distributed memory programming models, specifically focusing on techniques targeting message-passing applications using MPI due to its prevalence in HPC. Conclusion contains the lessons learned from our evaluation and concludes the article.

1. Concepts and Definitions

In this section, we introduce concepts and definitions that recur throughout the manuscript. Regardless of whether an R&R technique is being used for debugging, computational reproducibility, or fault-tolerance, the technique must observe and represent nondeterministic application behavior during the recording phase, and then recover that same behavior during replay. First, we introduce terms to describe the generic styles of R&R. Next, we define terms that are common across all R&R techniques. We conclude this section by introducing the concept of replay fidelity and relating it to the level of determinism present in the recorded application.

Data-replay techniques explicitly record the contents of communication. In the context of R&R techniques for distributed memory applications, specifically message-passing applications, a data-replay technique records the contents of messages. For techniques targeting shared-memory applications, a data-replay technique records the values read from or written to shared

memory locations. In contrast with data-replay, order-replay techniques only record the relative ordering of communication. In the context of distributed-memory applications, specifically message-passing applications, an order-replay technique records the interleaving of message receptions. For shared-memory applications, an order-replay technique records the order in which threads accessed shared memory locations. Order-replay techniques avoid the overheads associated to copying and storing large message contents, but must make more assumptions about the global ordering of communication (e.g., that messages are delivered through FIFO queues) than data-replay techniques. Hybrid-replay techniques combine data-replay and order-replay approaches in an attempt to balance the greater scalability of order-replay with the richness of debugging information and flexibility that data-replay provides (e.g., replaying only a subset of processes). Techniques that aim to avoid replaying many non-buggy processes in order to replay a single buggy one are referred to as subgroup-replay techniques.

Every R&R technique builds up a representation of an execution during the recording phase. This representation exists in memory during the recording and may be persisted to disk, either in chunks during recording or all at once at the end, and necessarily imposes some overhead in order to monitor events and update the representation. Consequently, we will refer to two kinds of overheads: memory overhead and execution time overhead. The memory overhead refers to the size of the in-memory representation, and the execution time overhead refers to the slowdown relative to an unmonitored execution. Additionally, we will refer to the on-disk representation as the trace of the execution. Each technique also defines a level of replay fidelity, by which we mean a contract-like description of which events in the replayed run are guaranteed to match those from the recorded run. For instance, a technique may guarantee that messages are received by each process in the same order during replay only if they are sent in the same order during replay. Another technique may guarantee that both types of events (i.e., receives and sends) interleave in the same order. In this case, the second technique is said to have higher replay fidelity than the first one.

Replay fidelity, specifically the ease with which the desired level of it can be achieved, is intimately related to the determinism class [7] of the application being recorded. HPC applications commonly consist of intervals of deterministic computation bounded by periods of communication or synchronization, which may incorporate some element of non-determinism. The determinism class of an application is largely determined by how this communication is implemented, specifically how it is overlapped by computation. Generally speaking, a high level of replay fidelity is more challenging to achieve for applications the determinism class of which is less strict.

2. Record-and-Replay for Shared Memory

In this section we discuss R&R techniques that target shared memory models. These techniques target replaying the behavior of an application on a single node of an HPC system. The presence of fat nodes and accelerators in HPC systems makes these techniques relevant for peta and exascale computing. First, we discuss early approaches to R&R for shared memory applications. Next, we cover techniques that employed logical clocks as a means of managing trace growth as systems scaled in size. Finally, we survey contemporary techniques which we classify as either debugging-oriented (i.e., emphasizing high-fidelity replay) or targeting reproducibility of execution characteristics in a more relaxed sense.

2.1. Early Approaches – Plurality of Directions

From the late 1980s through the 1990s, R&R for shared memory models explored tradeoffs between replay styles (i.e., data-replay vs. order-replay), but focused primarily on debugging, and thus, high-fidelity replay. In 1987, LeBlanc introduced Instant Replay [23], which adopted an order-replay approach of associating a version number to each shared-memory access. Instant Replay is a common predecessor not only of later order-replay techniques for shared memory models, but also of many techniques for distributed memory models as discussed in Section 3. The empirical evaluation of Instant Replay was conducted on a 128-cpu system and used a Gaussian Elimination workload, indicating early interest in facilitating debugging of HPC applications. In contrast, Pan's 1988 Recap [38] technique combined periodic checkpointing with a data-replay logging approach. Additionally, Recap requires the compiler to generate instrumentation code, rather than being implemented as a library. While achieving greater flexibility of replay (e.g., the ability to replay from a checkpoint as opposed to the start of the execution), Recap is limited by the daunting growth rate of its log. Beyond order and data-replay, hardware-assisted approaches also emerged. The first R&R technique designed to explicitly take advantage of hardware was introduced by Bacon and Goldstein in 1991 [3]. The latter technique logs traffic between the main memory and caches and establishes a total order on shared-memory accesses by recording instruction counter values.

In the early 1990s, Netzer introduced the shared memory version of his Optimal Replay technique [35], addressing the overhead associated with tracing every shared memory access via on-the-fly race detection using vector clocks [12]. Netzer's work on Optimal Replay, both the shared memory version discussed in this section and the distributed memory version discussed in the following section, are among the first in the R&R space to acknowledge the need for adaptive tracing, to explore the tradeoff between execution time overhead and the memory footprint of the trace [33], and to leverage logical clocks. In both the original and subsequent work on Optimal Replay, the technique was evaluated against the SPLASH-2 benchmark suite on up to 16 threads.

To contrast with the overheads imposed by using vector clocks in [35], Levrouw et al. introduced an order-replay technique [26] based on scalar logical clocks [22]. The authors provide theoretical justification and empirical evaluation supporting their claim that their technique can outperform its two major competitors at the time–Netzer's Optimal Replay and LeBlanc's Instant Replay–both in terms of recording overhead and memory footprint. However, the evaluation workloads do not resemble HPC workloads (e.g., sorting) and are run on a maximum of four threads.

This period of development culminated in Ronsse and de Bosschere's RecPlay [45] debugging framework. This work uses multiple kinds of logical clocks: scalar clocks to order synchronization events during recording, and snooped matrix clocks [4] to do race detection during replay and abort when data races prohibit correct replay. RecPlay was evaluated against the SPLASH-2 benchmark suite, demonstrating worst-case execution time overhead during recording of 25.9%. As will be seen throughout the rest of this manuscript, the idea of augmenting R&R through logical clocks persists to the present day.

2.2. Broadened Scope: Adding Reproducibility

Contemporary approaches to R&R of shared-memory applications address a broad spectrum of reproducibility challenges. In this section, besides the traditional debugging-centric techniques, we discuss techniques that target more relaxed notions of reproducibility. Across these techniques we observe two trends: increased diversity of programming models targeted, and increased interest in hardware assisted approaches.

2.2.1. Reproducibility-Centric Techniques

Due to the overheads associated with high-fidelity replay, recent techniques targeting multicore systems have opted to relax the requirement of exactly replaying the recorded run. Instead, these techniques try to replay a run the output of which matches the recorded run's output, regardless of the internal state-transitions that produce the output. Altekar introduced Output-Deterministic Replay (ODR) [2], which records only a subset of the necessary information to guarantee reproducible outputs but, augments this subset during replay with a search procedure to iterate towards a thread schedule that reproduces the final state of the recorded run. While this strategy can be applied to debugging, it can also clearly aid in situations where reproducibility of numerical outputs is desired. ODR was evaluated against a suite of applications including three of the SPLASH-2 benchmarks and demonstrated average recording overhead of 1.6×.

Simultaneously, Park introduced a similar technique called Probabilistic Replay with Execution Sketching (PRES) [39]. PRES also relaxes the common replay fidelity requirement, but in a different way. Rather than defining an equivalent execution more loosely, PRES proposes to gradually approach exact replay over multiple replay runs. By building up an execution sketch during recording and then iteratively exploring the constrained set of possible executions that agree with the sketch, PRES is able to hone in on executions that reproduce a buggy run. The evaluation was conducted on a subset of the SPLASH-2 benchmarks on up to four threads, at which scale execution time overhead of 10.6% was observed for an N-Body workload.

Inspired by fault-tolerance, Lee introduced Respec [24] which also targets determinism at the output level. In contrast with most popular techniques used at this point in time, Respec runs its "replay" concurrently with the monitored execution, periodically checking for divergence between the two runs and maintaining a checkpoint of the executions' last agreed-upon state. Respec manages overhead by only logging a subset of the most common synchronization operations, and borrows ideas from fault-tolerance protocols to correct itself when the replaying execution drifts too far from the monitored execution. Respec was evaluated, on up to four threads, on the PARSEC and SPLASH-2 benchmark suites.

At the beginning of this decade, many codes are parallelized on HPC systems by using a shared-memory model on each node coupled with a distributed-memory model for internode communication (i.e., the MPI/X model). Budanur et al., introduced an R&R technique for studying memory inefficiencies in this scenario called ScalaMemTrace [6]. ScalaMemTrace leverages Extended Power Regular Section Descriptors (EPSRDs) to exploit repeated behavioral patterns across multiple levels of the memory hierarchy to realize a compressed representation of a memory trace. ScalaMemTrace was evaluated against HPC-centric workloads, namely the MPI/OpenMP implementation of the Sequoia AMG benchmark and two computational kernels: matrix-matrix multiplication and vector addition. In a weak-scaling study of the kernels, ScalaMemTrace's compression afforded near-constant trace size on up to 64 threads, in contrast

with the uncompressed version of the same trace which grew exponentially with the number of threads. In a strong-scaling study of the AMG benchmark, the compressed traces grew linearly with the number of threads, but nevertheless afforded a 50% reduction in size. The manuscript acknowledges that ScalaMemTrace's replay fidelity is limited, namely that the AMG benchmark was only replayed at 91% accuracy.

2.2.2. Debugging-Centric Techniques

While the first reproducibility-centric R&R techniques appeared and evolved, as described in the previous section, debugging-centric techniques still remain a high priority. The major challenge shared memory presents for debugging is the sheer number of events (i.e., shared memory accesses) that must be traced to ensure the high-fidelity replay required. Contemporary debugging-centric techniques address this challenge via two strategies: first, working primarily at the software level (e.g., by employing an algorithm to ensure that a necessary and sufficient trace is recorded for the desired level of replay fidelity); and second, by leveraging specialized hardware to reduce recording overhead (e.g., by using hardware clocks with tighter synchronization guarantees).

At the software level, we observe a single vendor-driven effort to address the challenges of R&R with a highly general technique. Intel proposed PinPlay [40], which is an R&R technique built atop their Pin dynamic instrumentation framework. PinPlay provides highly flexible R&R options (e.g., subgroup replay) and offers composability with other Pin-based tools (thus, pursuing generality). PinPlay's evaluation was conducted on realistic HPC workloads, including the Parallel Ocean Program, the MILC quantum chromodynamics code, and a weather prediction application. On these workloads, PinPlay observes between 36× and 147× execution time overhead during recording, which, despite being high, is indicative of the challenge of recording sufficient information for high-fidelity replay of scientific applications.

From the academic community, efforts include reformulation of replay as a satisfiability problem, and specialization to task-parallel runtimes.

Liu et al.'s Light [28] introduces a novel software-level technique leveraging Satisfiability Modulo Theory (SMT) solvers. Light formally characterizes the minimal trace data needed for the desired level of replay. Light focuses on logging flow dependence of shared memory accesses, and then during replay generates a thread schedule that respects the recorded flow dependencies by formulating it as a satisfiability problem. Light is evaluated on a diverse set of benchmarks ranging from scientific applications to web server and web crawling applications.

Though all techniques in this section have focused on recording threads' behavior, in the case of task-based parallel runtimes such as Cilk Plus, this is not an option since the runtime's scheduler may employ variable numbers of threads from run to run in a nondeterministic fashion. A so-called "Processor-oblivious" R&R is needed in this case. PORRidge [51] is the first such technique of this kind, which targets Cilk Plus programs where shared objects are protected by locks. Rather than maintaining per-thread records, PORRidge maintains per-lock records and constrains the Cilk scheduler to conform to these records' access orders by augmenting the scheduler's DAG-representation of the execution with extra edges that represent the happens-before relationships observed during recording. PORRidge is evaluated against a diverse set of benchmarks, including two from the PARSEC suite, and three nondeterministic graph algorithms from the Problem Based Benchmark Suite. The execution time overhead of the recording phase

ranges from essentially negligible to 3.39×, with a mean of 1.62× across all benchmarks. The graph benchmarks saw the highest overheads for both recording and replaying.

In parallel with novel software-level approaches to R&R, researchers have endeavored to exploit advances in hardware and propose future hardware augmentations. In an effort to reduce recording overhead while maintaining the level of replay fidelity needed for debugging Hower et al. introduced Rerun [17], which proposes to use relatively little dedicated memory per core compared with prior approaches, coupled with a scalar logical clock to establish a partial order on atomic episodes, i.e. periods during which one thread does not race with any other. Additional hardware-assisted techniques were proposed throughout the 2010s, most notably QuickRec [41] which proposes an extension of Intel's architecture to support efficient record and replay, and SAMSARA [44] which leverages hardware-assisted virtualization extensions. QuickRec was evaluated on the SPLASH-2 benchmarks on up to eight threads, whereas SAMSARA was evaluated on the PARSEC benchmarks on up to four threads.

More recently, the tool Castor [30] has been developed to provide recording with such low overhead that it may be left on by default, rather than only used during specific debugging runs. Castor uses a time stamping procedure that relies on synchronized hardware clocks. In contrast with other tools, Castor monitors replayed execution and dynamically evaluates whether the replay sufficiently matches the record. Castor maintains per-thread records of nondeterministic events which are aggregated to a master record periodically. Castor's primary contribution is maximization of log throughput during recording and replay by use of transactional memory and recording nondeterminism at multiple levels by combining an LLVM pass, library interpositioning, and thread-level logging. Castor is evaluated against the PARSEC benchmark suite. Though Castor demonstrates low recording overhead for most PARSEC benchmarks, the Radiosity benchmark incurs overheads up to 25% when running on 10 threads. The authors claim that this is due to Radiosity's sensitivity to the log aggregation's effect on caches and the fact that Radiosity periodically overwhelmed the aggregator.

3. Record-and-Replay for Distributed Memory

In this section we discuss R&R techniques that target distributed memory models. As the Message-Passing Interface (MPI) [32] became the *de facto* standard for distributed-memory HPC applications, R&R techniques increasingly targeted MPI applications. Therefore, we focus on these techniques in this section. First, we discuss early approaches to R&R techniques, moving from data-replay to order-replay in pursuit of scalability. Next, we identify an era of progress in order-replay due in large part to the integration of logical clock algorithms. Finally, we identify contemporary techniques which we classify into three main research directions: first, debugging-oriented techniques with an emphasis on high-fidelity replay; second, techniques oriented towards reproducibility of communication characteristics and performance analysis; and third, techniques that straddle the line between R&R and fault-tolerance.

3.1. Early Approaches – from Data-Replay to Order-Replay

The earliest era of R&R techniques for distributed memory applications took the data-replay approach for three reasons. First, early R&R tools were developed in the debugging community rather than the HPC community, so logging messages' content to provide rich debugging information at the expense of scalability was seen as an acceptable tradeoff. Second, message-passing

standards such as MPI were still maturing, which limited the development of order-replay techniques. Third, data-replay does not require that all processes be replayed, allowing users to focus on buggy processes. Representative data-replay techniques from this era include BUGNET [11] and Recap [38]. BUGNET logged all message contents and represents one of the first tools specifically designed to alleviate the burden of nondeterminism when debugging parallel code. Recap represented a step forward in two ways. First, and most prominently, Recap was one of the first tools to provide subgroup replay; recognizing that in the debugging context, only faulty processes need to be replayed. Second, Recap explicitly acknowledges that the rate of trace generation constrains the technique's applicability.

As the HPC community began to recognize the need for R&R tools, order-replay soon emerged as an attractive alternative to data-replay. LeBlanc's Instant Replay [23] was the earliest such technique, as discussed in Section 2.1. Instant Replay models all interprocess communication as accesses to shared objects and assigns version numbers to each such access during the recording phase. This approach circumvents the need to copy and store message buffers' contents, and so is attractive for the debugging scenario where an application may need to run for a long time before a bug manifests. The empirical evaluation of Instant Replay demonstrated the space savings that can be realized by an order-replay approach. Specifically, LeBlanc shows that for a Gaussian Elimination workload on 64 processes, Instant Replay's trace occupied over 300x less space than a data-replay trace wherein all messages' contents are logged.

Despite achieving significant reduction in trace size, Instant Replay presented challenges to practical implementation, which were identified and addressed by Leu et al. in their proposal of Execution Replay [25]. Leu observed that Instant Replay's versioning algorithm requires sending extra messages, which may interfere with debugging. Execution Replay constrains itself to send no additional messages. Additionally Leu took steps towards formalizing the notion of equivalent executions (i.e., What does it mean to say that a replay is "correct"?). Finally, Leu explicitly addresses replay of both blocking and non-blocking communication–a distinction absent in prior work. Throughout this initial period of development, we observe the emergence of the key driving problems that guide research into R&R to this day (i.e., the tradeoff between memory overhead and execution time overhead). Moreover, the evolution of systems architecture exacerbated the need to manage the above-mentioned tradeoff. This lead to the burst of research on R&R techniques throughout the 1990s and early 2000s discussed in the next section.

3.2. Coping with Scale for Debugging

Order-replay techniques were developed in response to the growing need in HPC for scalable R&R. Through the 1990's and early 2000's, two related design goals emerged. First, minimizing the memory overhead and trace size was recognized as necessary to apply R&R to long-running, memory-hungry HPC applications. Second, minimizing the execution time overhead was recognized as necessary, because even small changes in event timings could prevent bugs from manifesting during the recording phase. Logical clocks [22] (i.e., algorithms for establishing orders on events across multiple processes) became instrumental to tackling both problems. Both challenges are addressed in the following sections, discussing enhancments associated to logical clocks and hybridization of replay techniques.

3.2.1. Enhancing Order-Replay Techniques with Logical Clocks

Netzer's work on Optimal Replay [36] is archetypical of the research direction for R&R during the 1990's. The general idea was to detect potential message races at runtime using a vector-valued logical clock [12] and log only enough information to cause the races' outcomes during replay to match their outcomes during the recorded run. All other non-racing communication could be safely assumed to repeat without the guidance of the recorded trace. Empirical evaluation of Optimal Replay on a set of computational kernels (e.g., matrix multiplication) indicated that only 1-14% of messages required tracing, leading to significant reductions in tracing overhead. Beyond Optimal Replay, Netzer also introduced Incremental Replay [34], which combined checkpointing with the kind of message-logging common in R&R to allow replayed executions to "fast-forward" to the region of the execution where a bug actually occured. While the focus on debugging is clear, this work is indicative of the conceptual overlaps between the debugging and HPC fault tolerance communities – a trend that has persisted to the present.

Shortly following Netzer's work on Optimal Replay, MPI was standardized and embraced by the HPC community as the *de facto* mechanism for distributed memory scientific applications. Consequently, R&R techniques for distributed memory applications began to specifically target MPI. The earliest example of this is Clemencon's work [9] on the *Annai* programming environment's parallel debugging tool. Clemencon's technique builds on the vector-clock-based race detector of Optimal Replay, adding an additional scalar logical clock per process. This work is significant not only in its use of logical clocks, but also in that it addresses the nondeterminism inherent in non-blocking communication, including non-blocking probes. This work was evaluated on communication microbenchmarks, as well as more HPC-oriented benchmarks such as BiCGSTAB, on up to 64 processes.

Despite the power of vector clocks to determine when events must be recorded, the need to piggyback vector timestamps on all messages becomes increasingly prohibitive as system size increases. ROLT^{MP} [46] attempts to overcome this scalability barrier by using scalar logical clocks during recording to determine when events must be explicitly logged, coupled with additional checks during replay to compensate for the limited ability of scalar clocks to order events across processes [8].

As MPI saw increasingly widespread adoption in HPC, techniques for R&R emerged that leverage MPI's point-to-point communication semantics in order to reduce recording overhead for applications with limited nondeterminism. Kranzlmuller introduced such a technique in the Nondeterministic Program Evaluator (NOPE) [20]. This work was novel not only in that it forgoes logical clocks in favor of a simpler logging mechanism that leverages MPI's non-overtaking rule, but also in that NOPE supports replay of alternative executions (i.e., one execution is recorded but multiple possible executions, including the recorded one, may be replayed). NOPE's drawbacks are that it assumes the only sources of receiver-side nondeterminism are wildcard receives, and that the recorded application is send-deterministic [7].

In addition to leveraging the features of MPI itself, R&R tools were developed in response to the particular programming idioms HPC developers wrote into their codes. De Kergommeaux et al. developed MPL* [18] which targeted correct replay of MPI applications that use non-blocking test functions in polling loops. This programming idiom is commonly used to overlap communication and computation, but one result is that the number of tests becomes nondeterministic. MPL* compactly represents sequences of test calls such that applications that use the number of test calls (e.g., for control flow) can be correctly replayed. Kranzlmuller later went a step

beyond, opting to integrate R&R functionality into the MPI library itself [19]. The primary advantage of such a scheme is convenience for the user in a debugging context since there is no need to instrument code, link with additional libraries beyond MPI, or manage traces. Later work by Bouteiller et al. produced Retrospect [5], which integrates R&R directly into an MPI implementation's lower-level communication routines and can switch between data-replay and order-replay strategies based on user needs. Retrospect set a new standard for empirical evaluation of distributed memory R&R techniques as it was evaluated on the NAS Parallel Benchmarks on up to $1,024$ processes.

3.2.2. Hybrid Replay and Integration of Checkpointing

Due to the growing popularity of order-replay, data-replay approaches were not explored to the same degree. However, notable exceptions include Gertsel's On-the-Fly Replay [13] and several works of Zambonelli [55, 56]. Gerstel's work introduced a scheme where the entire network of processes is duplicated, and one-way channels between the processes in the "real" network propagate the nondeterministic actions of those processes to their partners in the other network. In a sense, this scheme pursues "replay-while-recording" debugging. In contrast, Zambonelli's work builds off of Netzer's work on Incremental Replay, first providing a solution to avoid deadlocks then optimizing its message logging algorithm. This work showed that by piggybacking limited additional information on all messages, its message-logging scheme reduced the number of messages that needed to be logged by 10%.

Along similar lines, Thoai's Shortcut Replay [50] proposes replaying from a combination of globally consistent and inconsistent checkpoints. The value proposition of Shortcut Replay is being able to skip segments of the execution during replay that are not immediately relevant to a debugging task. Shortcut Replay leverages the event graph model of MPI programs' executions to determine how to safely combine checkpoints to avoid inconsistency during replay.

3.3. Broadened Scope: Adding Reproducibility and Fault Tolerance

Contemporary research in R&R has advanced along two novel directions in addition to debugging: reproducibility and fault-tolerance. First, a class of R&R techniques targeting more relaxed definitions of reproducibility (e.g., for characterizing the communication behaviors of applications or studying performance portability) has emerged. These techniques do not provide the fine-grained, high-fidelity replay of debugging-oriented techniques, but instead prioritize scaling to larger executions. Second, cross-pollination between the debugging and fault-tolerance communities has led to a class of fault-tolerance protocols involving message-logging that leverage techniques from traditional R&R. First we discuss these new directions, and we conclude this section by discussing contemporary approaches to debugging-centric record-and-replay.

3.3.1. Reproducibility-Centric Techniques

Although debugging continues to be a major focus of R&R, recent interest from the HPC community in reproducibility (of e.g., performance, communication behavior, scientific outcomes of simulations) has motivated R&R techniques that target replay with a more relaxed fidelty requirement than in the debugging case.

A prominent example of this trend is the Scalatrace tools. The original Scalatrace [37] tool proposed trace compression techniques that provide nearly constant-sized traces for MPI

applications with sufficiently regular communication. Regular section descriptors (RSDs) and their recursive counterparts power-RSDs are used to compactly represent MPI communication events, which, combined with a suite of MPI-specific encoding techniques, significantly reduce trace sizes for applications with relatively little nondeterminism.

Scalatrace's immediate successor, Scala-H-Trace [53] provides probabilistic replay. Rather than exactly replicating fine-grained MPI events such as individual message receptions, Scala-H-Trace allows the user to specify a tuning parameter that controls the tradeoff between fidelity of replay and scalability of recording. By forgoing the lossless tracing approach of its predecessor, Scala-H-Trace was able to achieve near-constant trace file sizes when recording executions of the Parallel Ocean Program on up to $2,048$ processes. In contrast, the original Scalatrace was only able to record the same executions on up to $1,1024$ processes. Additionally, Scala-H-Trace demonstrated constant trace sizes for the NPB conjugate gradient workload on $2,048$ processes. An important caveat is that these results were obtained using the lowest replay fidelity setting of Scala-H-Trace, and the metric used for evaluating correctness of replay is overall execution time. Despite limited applicability to debugging, techniques providing probabilistic replay can be invaluable for scientific reproducibility and for ensuring that the communication behavior of applications has characteristics that agree with programmer intent.

Scala-H-Trace's successor, Scalatrace II [52], also embraces probabilistic replay. Proposed as a major redesign of the compression techniques developed in Scalatrace for applications with more irregular behavior, Scalatrace II is evaluated against a subset of NPB, the Sweep3D neutron transport kernel, and the Parallel Ocean Program on up to 400 processes. Like Scala-H-Trace, Scalatrace II performs lossy compression of MPI events and thus its correctness metric for replay is overall execution time. In all evaluation workloads, Scalatrace II achieves good agreement between replay time and originally observed execution time–experience an average execution time error of 5.7%. Like its predecessor, the probabilistic replay Scalatrace II provides is useful for achieving coarse-grained reproducibility of communication behavior.

Scalatrace was evaluated on stencils of various sizes, a subset of the NPB, and Raptor, a simulation framework supporting adaptive mesh refinement used in this evaluation to run a hydrodynamics simulation. These workloads were recorded on up to 512 processes and in most cases demonstrated nearly constant trace size and memory footprint during recording. However, the conjugate gradient and Fourier transform benchmarks from NPB yielded trace sizes that increased with the number of the processors. In the case of the conjugate gradient benchmark, the role of asynchronous point-to-point communication in the increased trace size is acknowledged.

Most recently, Zhai et al. [57] introduced "Representative Replay", which targets performance prediction. Representative replay permits users to record an execution of a parallel application on a limited platform (e.g., a single node of a cluster) and obtain predictions of the execution time of the same application on the full platform via a controlled replayed execution.

3.3.2. Fault-Tolerance-Centric Techniques

The last aspect of R&R's applicability is in fault-tolerance. As the exascale era looms, fault-tolerance occupies an increasingly important role in HPC. R&R techhniques and fault-tolerance protocols (especially their message-logging aspects) share many concerns, namely concise representation of nondeterministic application behaviors. Similarities can also be seen between debugging-oriented techniques that seek to replay only buggy processes, and fault-tolerance protocols that combine checkpointing with message-logging; seeking to avoid rolling back ex-

ecution any farther than necessary. In this section, we discuss fault-tolerance protocols that incorporate deterministic replay.

Partial message-logging protocols, such as those introduced by Guermouche et al. [16] and Meneses et al. [31] depend on clustering of processes into process groups that realize similar communication patterns in order to achieive efficient deterministic replay. Ropars et al. introduced a process clustering technique [47] that leverages the regular communication patterns of MPI collectives and uses a bisection-based graph partitioning algorithm to compute process clusters that facilitate partial message-logging protocols. For empirical evaluation, Ropars' clustering algorithm is coupled with the above-cited partial message-logging protocols and ran against set an HPC benchmarks including the NAS Parallel Benchmarks and LAMMPS. The key contribution of this work is that the process clustering algorithm takes the characteristics of the partial message-logging scheme it is coupled to into account, resulting in process clusters that are conducive to reducing the overall number of messages logged. Specifically, the technique requires logging less than 5.3% of all messages for all but one of their workloads.

Due to the daunting task of achieving fault-tolerance at exascale, reseachers seek to leverage application characteristics to achieve scalable replay of failed processes. Lifflander et al. developed an algebraic methodology for their fault-tolerance protocol [27] that can determine when events in the execution of a distributed-memory application may safely commute (i.e., occur in any order without altering the program state after their completion), thus a message-logging R&R scheme with lower overheads. Beyond this work's theoretical contributions, the fault-tolerance protocol was rigourously evaluated against HPC applications across a variety of scientific domains (e.g., molecular dynamics and hydrodynamics). The evaluation platform was an IBM BG/P system with $40,960$ nodes, and the workloads were ran on up to $131,072$ processes.

3.3.3. Debugging-Centric Techniques

The debugging-oriented research direction of R&R continues to prioritize reducing the trace size and memory footprint of the recording phase. Xue et al. proposed Subgroup Reproducible Replay (SRR) [54] to address not only the problem of trace size, but also provide the user with flexible replay options (i.e. replaying only those processes deemed relevant to the debugging task). SRR leverages the fact that most HPC applications are structured so that any one process does most of its communication with a limited subset of other processes (e.g., communicating only with its neighbors in a stencil pattern). With this in mind, SRR adopts a hybrid-replay strategy by recording only message interleavings within groups of processes (order-replay), while recording the contents of messages between groups (data-replay). While it is acknowledged that application developers' expertise may be needed to specify the appropriate process groups, the grouping can also be obtained by partitioning a weighted graph representing the application's communication channels. SRR was evaluated on a diverse benchmark suite containing a subset of the NAS Parallel Benchmarks, as well as a parallel graph benchmark and a benchmark specifically designed to stress SRR's ability to replay nondeterministic probes. The benchmarks were run on a maximum of 64 processes.

In a distinct approch to hybrid-replay, Gioachin et al. introduced "Processor Extraction" [14] where multiple R&R passes are used to iteratively progress towards replaying a desired bug. The first pass uses a lightweight order-replay approach. Subsequent passes use increasingly heavyweight data-replay approaches but on increasingly small subsets of processes, thereby

narrowing down the root cause of bugs and limiting the tracing overhead. This technique was evaluated on up to $1,024$ processes. The evaluation workloads ranged from synthetic benchmarks to full-fledged scientific applications, specifically ChaNGa and NAMD.

Despite the flexibility of hybrid-replay schemes, pure order-replay remains attractive for HPC, especially when fine-grained high-fidelty replay is desired for debugging. To this end, Sato et al. developed Clock-Delta Compression (CDC) [49], a technique for compressing the record of nondeterministic events. The technique piggybacks scalar logical timestamps on each message much in the same way that earlier techniques such as ROLT^{MP} do, but instead of explicitly storing the timestamps, CDC stores a permutation that maps a reference order, which can be deterministically generated during replay, to the observed order from the recorded run, thereby realizing a reduction in trace size. Clock-Delta Compression was evaluated against an HPC proxy application "The Monte Carlo Benchmark" (MCB) [10] on $3,072$ processes, for which CDC was able to show a reduction in trace size of two orders of magnitude while maintaining execution time overhead of 13.1-25.5%.

Finally, as MPI continues to evolve, so must the R&R techniques targeting it. MPI's one-sided communication routines in particular pose unique challenges to R&R. Quian et al. proposed two techniques for addressing this challenge–OPR [42] and its successor SReplay [43]. SReplay proposes a hybrid-replay scheme which permits replay of subgroups of processes. Like MPIWiz, SReplay does order-replay within process groups for which it employs a specialized vector clock algorithm. Unlike previous works that have associated vector clocks to individual memory regions, ranges of regions are associated to the same vector clock in SReplay thereby enhancing its scalability. SReplay was evaluated against standard benchmarks for R&R tools (e.g., the NAS Parallel Benchmarks), but also against applications exhibiting considerably greater nondeterminism (e.g., unbalanced tree search and parallel DeBruijn graph construction) on up to $1,024$ processes.

Conclusion

Record-and-replay techniques were originally developed enabling cyclic debugging in the presence of nondeterminism, but since then their scope has expanded to include more relaxed forms of reproducibility, as well as integrating into fault tolerance protocols. This expansion of scope is driven by the three communities we identify as associated to R&R. In this section we revisit the guiding questions of each community listed in Section and provide answers.

For researchers whose goal is to develop new R&R techniques, the guiding question is: Where are the gaps in the technique space? We observe three gaps. First, there are no techniques that offload recording overhead to accelerators, which are becoming more prevalent on HPC systems. Second, debugging-oriented techniques have not been evaluated at the scale of full system runs on petascale platforms. Third, there has been minimal investigation of composability of shared memory and distributed memory techniques.

For HPC developers, the guiding question is: What are the workloads to which R&R techniques apply? We observe that the workloads R&R techniques are evaluated against have grown in complexity (as evidenced in Fig. 1 by the increase in full HPC application evaluation workloads with time), however the scale at which recording is feasible depends strongly on desired replay fidelity. For reproducibility studies, recording runs on over $5K$ processes has become feasible. For debugging, particularly of long-running applications, $5K$ processes is still a barrier.

Finally, for exascale-oriented researchers, the guiding question is: What are the open problems preventing state-of-the-art R&R from being deployed at exascale? For this community, we consider first the ways R&R overcame scaling challenges in the past, then propose problems to consider for the future. Driven by the need to minimize recording overhead, R&R techniques employed logical clock algorithms, leveraged compression to reduce memory overhead and trace size, and proposed methods that adapt to application characteristics. For R&R to remain viable at exascale we observe three potential untapped directions. First, elements of recording and replaying may be able to be offloaded to GPUs or other accelerators. Second, machine learning may be needed to more-precisely adapt recording strategies to application characteristics. Third, logical clock algorithms continue to advance (e.g., logical-physical clocks [21]), exposing the possibility of upgrading old techniques with new event-ordering strategies.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by LLNL under contract DE-AC52-07NA27344 (LLNL-JRNL-749010).

This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

References

1. Ahn, D.H., Lee, G.L., Gopalakrishnan, G., Rakamarić, Z., Schulz, M., Laguna, I.: Overcoming extreme-scale reproducibility challenges through a unified, targeted, and multilevel toolset. In: Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering. pp. 41–44. SE-HPCCSE '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2532352.2532357

2. Altekar, G., Stoica, I.: ODR: Output-deterministic Replay for Multicore Debugging. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. pp. 193–206. SOSP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1629575.1629594

3. Bacon, D.F., Goldstein, S.C.: Hardware-assisted Replay of Multiprocessor Programs. In: Proceedings of the 1991 ACM/ONR Workshop on Parallel and Distributed Debugging. pp. 194–206. PADD '91, ACM, New York, NY, USA (1991), DOI: 10.1145/122759.122777

4. Bosschere, K.D., Ronsse, M.: Clock snooping and its application in on-the-fly data race detection. In: 1997 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '97), 18-20 December 1997, Taipei, Taiwan. pp. 324–330 (1997), DOI: 10.1109/ISPAN.1997.645115

5. Bouteiller, A., Bosilca, G., Dongarra, J.: Retrospect: Deterministic Replay of MPI Applications for Interactive Distributed Debugging. In: Proceedings of the 14th European PVM/MPI User's Group Meeting, Paris, France, September 30 – October 3, 2007. pp. 297–306. Springer, Berlin, Heidelberg (2007), DOI: 10.1007/978-3-540-75416-9_41

6. Budanur, S., Mueller, F., Gamblin, T.: Memory trace compression and replay for SPMD systems using extended PRSDs. SIGMETRICS Performance Evaluation Review 38(4), 30–36 (2011), DOI: 10.1145/1964218.1964224

7. Cappello, F., Guermouche, A., Snir, M.: On communication determinism in parallel HPC applications. In: Proceedings of the 19th International Conference on Computer Communications and Networks, IEEE ICCCN 2010, Zürich, Switzerland, August 2-5, 2010. pp. 1–8 (2010), DOI: 10.1109/ICCCN.2010.5560143

8. Charron-Bost, B.: Concerning the size of logical clocks in distributed systems. Information Processing Letters 39(1), 11–16 (1991), DOI: 10.1016/0020-0190(91)90055-M

9. Clémençon, C., Fritscher, J., Meehan, M.J., Rühl, R.: An implementation of race detection and deterministic replay with MPI, pp. 155–166. Springer, Berlin, Heidelberg (1995), DOI: 10.1007/BFb0020462

10. Cleveland, M.A., Brunner, T.A., Gentile, N.A., Keasler, J.A.: Obtaining identical results with double precision global accuracy on different numbers of processors in parallel particle Monte Carlo simulations. Journal of Computational Physics 251, 223–236 (2013), DOI: 10.1016/j.jcp.2013.05.041

11. Curtis, R., Wittie, L.D.: BUGNET: A debugging system for parallel programming environments. In: Proceedings of the 3rd International Conference on Distributed Computing Systems, Miami/Ft. Lauderdale, Florida, USA, October 18-22, 1982. pp. 394–400 (1982)

12. Fidge, C.J.: Partial orders for parallel debugging. In: Proceedings of the ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging, University of Wisconsin, Madison, Wisconsin, USA, May 5-6, 1988. pp. 183–194 (1988), DOI: 10.1145/68210.69233

13. Gerstel, O.O., Zaks, S., Hurfin, M., Plouzeau, N., Raynal, M.: On-the-fly replay: a practical paradigm and its implementation for distributed debugging. In: Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing, SPDP 1994, Dallas, Texas, USA, October 26-29, 1994. pp. 266–272 (1994), DOI: 10.1109/SPDP.1994.346158

14. Gioachin, F., Zheng, G., Kalé, L.V.: Robust Non-intrusive Record-replay with Processor Extraction. In: Proceedings of the 8th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging. pp. 9–19. PADTAD '10, ACM, New York, NY, USA (2010), DOI: 10.1145/1866210.1866211

15. Gopalakrishnan, G., Hovland, P.D., Iancu, C., Krishnamoorthy, S., Laguna, I., Lethin, R.A., Sen, K., Siegel, S.F., Solar-Lezama, A.: Report of the HPC correctness summit, January 25-26, 2017, Washington, DC. CoRR abs/1705.07478 (2017), http://arxiv.org/abs/1705.07478, accessed: 2017-12-22

16. Guermouche, A., Ropars, T., Brunet, E., Snir, M., Cappello, F.: Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic MPI Applications. In: Proceedings of the 25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May 2011. pp. 989–1000 (2011), DOI: 10.1109/IPDPS.2011.95

17. Hower, D., Hill, M.D.: Rerun: Exploiting Episodes for Lightweight Memory Race Recording. In: 35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China. pp. 265–276 (2008), DOI: 10.1109/ISCA.2008.26

18. de Kergommeaux, J.C., Ronsse, M., De Bosschere, K.: MPL: Efficient Record/Replay of nondeterministic features of message passing libraries, pp. 141–148. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/3-540-48158-3_18

19. Kranzlmüller, D., Schaubschläger, C., Volkert, J.: An Integrated Record&Replay Mechanism for Nondeterministic Message Passing Programs, pp. 192–200. Springer, Berlin, Heidelberg (2001), DOI: 10.1007/3-540-45417-9_28

20. Kranzlmüller, D., Volkert, J.: NOPE: A Nondeterministic Program Evaluator, pp. 490–499. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/3-540-49164-3_47

21. Kulkarni, S.S., Demirbas, M., Madappa, D., Avva, B., Leone, M.: Logical physical clocks. In: Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings. pp. 17–32 (2014), DOI: 10.1007/978-3-319-14472-6_2

22. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978), DOI: 10.1145/359545.359563

23. LeBlanc, T.J., Mellor-Crummey, J.M.: Debugging Parallel Programs with Instant Replay. IEEE Transactions on Computers 36(4), 471–482 (1987), DOI: 10.1109/TC.1987.1676929

24. Lee, D., Wester, B., Veeraraghavan, K., Narayanasamy, S., Chen, P.M., Flinn, J.: Respec: Efficient Online Multiprocessor Replay via Speculation and External Determinism. In: Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems. pp. 77–90. ASPLOS XV, ACM, New York, NY, USA (2010), DOI: 10.1145/1736020.1736031

25. Leu, E., Schiper, A., Zramdini, A.W.: Execution Replay on Distributed Memory Architectures. In: Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing, SPDP 1990, Dallas, Texas, USA, December 9-13, 1990. pp. 106–112 (1990), DOI: 10.1109/SPDP.1990.143516

26. Levrouw, L., Audenaert, K.M.R., Campenhout, J.M.V.: A New Trace And Replay System For Shared Memory Programs Based On Lamport Clocks. In: Proceedings of the Second Euromicro Workshop on Parallel and Distributed Processing, PDP 1994, January 26-28, 1994, Malaga, Spain. pp. 471–478 (1994), DOI: 10.1109/EMPDP.1994.592529

27. Lifflander, J., Meneses, E., Menon, H., Miller, P., Krishnamoorthy, S., Kalé, L.V.: Scalable replay with partial-order dependencies for message-logging fault tolerance. In: 2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014. pp. 19–28 (2014), DOI: 10.1109/CLUSTER.2014.6968739

28. Liu, P., Zhang, X., Tripp, O., Zheng, Y.: Light: Replay via Tightly Bounded Recording. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 55–64. PLDI '15, ACM, New York, NY, USA (2015), DOI: 10.1145/2737924.2738001

29. Lusk, E.L., Pieper, S.C., Butler, R.M., Univ., M.T.S.: More scalability, less pain : A simple programming model and its implementation for extreme computing. SciDAC Rev. 17(1), 30–37 (2010)

30. Mashtizadeh, A.J., Garfinkel, T., Terei, D., Mazieres, D., Rosenblum, M.: Towards Practical Default-On Multi-Core Record/Replay. In: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 693–708. ASPLOS '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3037697.3037751

31. Meneses, E., Mendes, C.L., Kalé, L.V.: Team-Based Message Logging: Preliminary Results. In: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia. pp. 697–702 (2010), DOI: 10.1109/CCGRID.2010.110

32. MPI: A Message-Passing Interface Standard, Version 3.0, http://mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf, accessed: 2017-12-22

33. Netzer, R.H.B.: Trace size vs parallelism in trace-and-replay debugging of shared-memory programs, pp. 617–632. Springer, Berlin, Heidelberg (1993), DOI: 10.1007/3-540-57659-2_35

34. Netzer, R.H.B., Xu, J.: Adaptive Message Logging for Incremental Replay of Message-passing Programs. In: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing. pp. 840–849. Supercomputing '93, ACM, New York, NY, USA (1993), DOI: 10.1145/169627.169850

35. Netzer, R.H.B.: Optimal Tracing and Replay for Debugging Shared-Memory Parallel Programs. In: Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, California, USA, May 17-18, 1993. pp. 1–11 (1993), DOI: 10.1145/174266.174268

36. Netzer, R.H.B., Miller, B.P.: Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs. In: Proceedings Supercomputing '92, Minneapolis, MN, USA, November 16-20, 1992. pp. 502–511 (1992), DOI: 10.1109/SUPERC.1992.236654

37. Noeth, M., Mueller, F., Schulz, M., de Supinski, B.R.: Scalable Compression and Replay of Communication Traces in Massively Parallel Environments. In: Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium. pp. 1–11 (2007), DOI: 10.1109/IPDPS.2007.370261

38. Pan, D.Z., Linton, M.A.: Supporting Reverse Execution for Parallel Programs. In: Proceedings of the 1988 ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging. pp. 124–129. PADD '88, ACM, New York, NY, USA (1988), DOI: 10.1145/68210.69227

39. Park, S., Zhou, Y., Xiong, W., Yin, Z., Kaushik, R., Lee, K.H., Lu, S.: PRES: Probabilistic Replay with Execution Sketching on Multiprocessors. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. pp. 177–192. SOSP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1629575.1629593

40. Patil, H., Pereira, C., Stallcup, M., Lueck, G., Cownie, J.: PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs. In: Proceedings of the

8th Annual IEEE/ACM International Symposium on Code Generation and Optimization. pp. 2–11. CGO '10, ACM, New York, NY, USA (2010), DOI: 10.1145/1772954.1772958

41. Pokam, G., Danne, K., Pereira, C., Kassa, R., Kranich, T., Hu, S., Gottschlich, J., Honarmand, N., Dautenhahn, N., King, S.T., Torrellas, J.: QuickRec: Prototyping an Intel Architecture Extension for Record and Replay of Multithreaded Programs. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. pp. 643–654. ISCA '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2485922.2485977

42. Qian, X., Sen, K., Hargrove, P., Iancu, C.: OPR: Deterministic Group Replay for One-sided Communication. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 47:1–47:2. PPoPP '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2851141.2851179

43. Qian, X., Sen, K., Hargrove, P., Iancu, C.: SReplay: Deterministic Sub-Group Replay for One-Sided Communication. In: Proceedings of the 2016 International Conference on Supercomputing. pp. 17:1–17:13. ICS '16, ACM, New York, NY, USA (2016), DOI: 10.1145/2925426.2926264

44. Ren, S., Li, C., Tan, L., Xiao, Z.: Samsara: Efficient Deterministic Replay with Hardware Virtualization Extensions. In: Proceedings of the 6th Asia-Pacific Workshop on Systems. pp. 9:1–9:7. APSys '15, ACM, New York, NY, USA (2015), DOI: 10.1145/2797022.2797028

45. Ronsse, M., De Bosschere, K.: RecPlay: A Fully Integrated Practical Record/Replay System. ACM Transactions on Computer Systems 17(2), 133–152 (1999), DOI: 10.1145/312203.312214

46. Ronsse, M., Kranzlmüller, D.: Rolt$^{\mathrm{mp}}$-replay of Lamport timestamps for message passing systems. In: PDP. pp. 87–93 (1998), DOI: 10.1109/EMPDP.1998.647184

47. Ropars, T., Guermouche, A., Uçar, B., Meneses, E., Kalé, L.V., Cappello, F.: On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications. In: Euro-Par 2011 Parallel Processing - 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part I. pp. 567–578 (2011), DOI: 10.1007/978-3-642-23400-2_53

48. Sato, K., Ahn, D.H., Laguna, I., Lee, G.L., Schulz, M., Chambreau, C.M.: Noise injection techniques to expose subtle and unintended message races. In: Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 89–101. PPoPP '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3018743.3018767

49. Sato, K., Ahn, D.H., Laguna, I., Lee, G.L., Schulz, M.: Clock delta compression for scalable order-replay of non-deterministic parallel applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015. pp. 62:1–62:12 (2015), DOI: 10.1145/2807591.2807642

50. Thoai, N., Kranzlmüller, D., Volkert, J.: Shortcut Replay: A Replay Technique for Debugging Long-Running Parallel Programs, pp. 34–46. Springer, Berlin, Heidelberg (2002), DOI: 10.1007/3-540-36184-7_5

51. Utterback, R., Agrawal, K., Lee, I.A., Kulkarni, M.: Processor-oblivious record and replay. In: Proceedings of the 22Nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 145–161. PPoPP '17, ACM, New York, NY, USA (2017), DOI: 10.1145/3018743.3018764

52. Wu, X., Mueller, F.: Elastic and Scalable Tracing and Accurate Replay of Non-deterministic Events. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. pp. 59–68. ICS '13, ACM, New York, NY, USA (2013), DOI: 10.1145/2464996.2465001

53. Wu, X., Vijayakumar, K., Mueller, F., Ma, X., Roth, P.C.: Probabilistic Communication and I/O Tracing with Deterministic Replay at Scale. In: International Conference on Parallel Processing, ICPP 2011, Taipei, Taiwan, September 13-16, 2011. pp. 196–205 (2011), DOI: 10.1109/ICPP.2011.50

54. Xue, R., Liu, X., Wu, M., Guo, Z., Chen, W., Zheng, W., Zhang, Z., Voelker, G.: MPIWiz: Subgroup Reproducible Replay of MPI Applications. In: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 251–260. PPoPP '09, ACM, New York, NY, USA (2009), DOI: 10.1145/1504176.1504213

55. Zambonelli, F.: Deadlock prevention in incremental replay of message-passing programs. In: Sloot, P., Bubak, M., Hoekstra, A., Hertzberger, B. (eds.) High-Performance Computing and Networking: 7th International Conference, HPCN Europe 1999 Amsterdam, The Netherlands, April 12-14, 1999 Proceedings. pp. 593–602. Springer, Berlin, Heidelberg (1999), DOI: 10.1007/BFb0100620

56. Zambonelli, F., Netzer, R.H.B.: Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999. pp. 392–398 (1999), DOI: 10.1109/IPPS.1999.760506

57. Zhai, J., Chen, W., Zheng, W., Li, K.: Performance Prediction for Large-Scale Parallel Applications Using Representative Replay. IEEE Transactions on Computers 65(7), 2184–2198 (2016), DOI: 10.1109/TC.2015.2479630