分类号： 密级：

UDC： 学号：

南 昌 大 学 硕 士 研 究 生

学 位 论 文

**基于RISC-V架构的容器化可重现方法研究**

**Research of Reproducible Container Based on**

**RISC-V Architecture**

崔傲

培养单位（院、系）： 数学与计算机学院

指导教师姓名、职称： 徐子晨 教授

申请学位的学科门类： 工学

学科专业名称： 计算机科学与技术

论文答辩日期：

答辩委员会主席：

评阅人：

年 月 日

文献综述

本文针对RISC-V上的程序执行的可重现性问题，设计实现一种轻量级的容器化可重现方法。可重现性可以进一步分解为两个子属性，即确定性和可移植性[10]：确定性保证相同输入情况下，反复执行程序始终得到相同的结果；可移植性则保证程序无需过多修改即可在不同机器上部署、执行。在软件工程中，通常使用确定性重放（Deterministic Replay，或记录重放，Record and Replay, R&R）技术实现程序的重现性。近年来，学术界对确定性重放技术相关研究高度重视，以下对其国内外研究现状进行综述。

基于软件的确定性从以下三个方向考虑

1、程序-确定性执行

可重现性的需求在19世纪80年代后期就引起了研究人员的注意。在软件调试过程中，软件开发人员通过循环执行顺序程序，发现和纠正程序执行中的错误。但是对并行程序的循环执行通常不会产生相同的结果。1987年，LeBlanc等人提出了*Instant Replay*[1]，作为一种重现并行程序执行行为的一般解决方案。*Instant Replay*通过记录重点事件发生的时序，保存程序重放所需的信息。*Instant Replay*后来成为了多项确定性重放技术的前身，乃至发展出了分布式内存上的可重现模型。

1. LeBlanc T J, Mellor-Crummey J M. Debugging parallel programs with instant replay[J]. IEEE Transactions on Computers, 1987, 36(4): 471-482.

1988年

确定性执行（Deterministic Execution）

2、环境-确定性执行系统

3、虚拟化-

2、环境-确定性执行系统

Aviram 等人2010年提出了*Determinator*，这是一个围绕确定性设计的操作系统抽象。*Determinator*实现了操作系统级别的确定性，强制单个进程，甚至交互进程组确定性执行。*Determinator*通过拒绝用户代码直接访问硬件资源来强制执行确定性，包括实时时钟、周期计数器和可写共享内存等。

1. Aviram A, Weng S C, Hu S, et al. Efficient {System-Enforced} Deterministic Parallelism[C]//9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10). 2010.

1 确定性重放研究现状

记录和回放(R&R)技术提供了监视和记录应用程序在一次执行(即记录的执行)中程序状态的变化，并重现这些变化，从而重现应用程序在后续执行(即重放执行)中的行为的能力。极高级别的硬件并发性、计算和通信的有效重叠以及总体代码复杂性使得R&R成为应对HPC应用程序中的非确定性的重要工具。HPC应用中的非确定性是一个日益严重的问题

1.1 共享内存的确定性重放

在本节中，我们讨论了目标共享内存模型的R＆R技术。这些技术的目标是在HPC系统的单个节点上重放应用程序的行为。HPC系统中的脂肪节点和加速器的实心，使得这些技术与Petaand ExaScale计算相关。首先，我们向R＆R讨论共享内存应用程序的早期方法。接下来，我们涵盖使用逻辑时钟作为管理TraceGrowth的手段的技术，因为系统缩放的系统缩放。最后，我们调查了我们Classifyas的当代技术，可以调试导向（即，强调高保真重播）或针对更轻松的意义上的执行特征的再现性。

从20世纪80年代末到20世纪90年代，R＆R为共享内存模型探讨了重播样式的平衡（即，数据重放与order-Replay），但主要关注调试，因此，高保真重播。 1987年，Leblanc介绍了即时重播[23]，该方法采用了将版本号与每个共享内存访问相关联的订单重播方法。 Instantreplay是一个常见的前任，不仅是共享的存储器模型的稍后订购重播技术，还具有分布式存储器模型的许多技术，如第3节所述的分布式存储器模型。在128-CPU系统上进行了即时重放的透明评估，并使用了高斯纽带工作量，表明早期兴趣促进HPC申请的调试。 INCONTRAST，PAN的1988 RECAP [38]技术结合了定期检查点，并通过数据重新计算方法进行了定期检查点。此外，RECAP要求编译器生成仪器处理代码，而不是被实现为库。在实现重放的更大灵活性（例如，从检查点重播的戏剧性而不是执行开始），RECAP受到其日志的令人生畏的增长率的限制。超越订单和数据重放，硬件辅助接近出现了。第一个R＆R技术，旨在明确地利用Bacon和Goldstein于1991年被培根和Goldstein的硬件[3]。后一种技术在主题内存和缓存之间日志流量，并通过记录仪计数器值建立共享内存访问的总顺序。在20世纪90年代初，Netzer介绍了他最佳ReplayTechnique [35]的共享内存版本，解决了与跟踪相关的开销每个共享内存访问VIEON-Fly竞争检测使用矢量时钟[12]。 Netzer在最佳重播中的工作，在本节中讨论的读数内存版本和分布式内存版本在以下部分中讨论，是R＆R空间中的第一个，以确认需要进行拓展跟踪，探讨执行时间开销和MemoryFootPrint之间的权衡迹线[33]，并利用逻辑时钟。在最佳重放的原始和后续工作中，该技术是针对Splash-2基准Suiteon评估最多16个线程的技术。与使用[35]中的矢量时钟，Levrouw等人的矢量钟表施加的开销对比。基于标量逻辑时钟[22]，介绍了订单重放技术[26]。提交人提供了解的理解和实证评估，支持他们的技术，他们的技术在Time-Netzer的最佳重播和Leblanc'Sinstant重播的两个主要竞争对手方面都在录制开销和内存足迹方面。但是，eval-uation工作负载不会类似于HPC工作负载（例如，排序），并且在最大的offour线程上运行。在ronsse和de bosschere的重复播放中有效的开发期间持续增长次数。这项工作采用多种逻辑时钟：标量时钟在录制期间订购同步性，而窥探矩阵时钟[4]在重放和禁止正确重放时，在重播和播放期间进行竞争检测。重新播放被评估对抗飞溅-2Benchmark套件，在整个稿件的剩余部分中展示25.9％的录制期间最坏情况的执行时间开销。

在20世纪90年代初，Netzer介绍了他最佳的重播技术[35]的共享内存版本，解决了与追踪每个共享内存访问veron-vere竞争检测的开销，使用矢量时钟[12]。 Netzer在最佳重播中的工作，在本节中讨论的读数内存版本和分布式内存版本在以下部分中讨论，是R＆R空间中的第一个，以确认需要进行拓展跟踪，探讨执行时间开销和MemoryFootPrint之间的权衡迹线[33]，并利用逻辑时钟。在最佳重放的原始和后续工作中，该技术是针对Splash-2基准Suiteon评估最多16个线程的技术。与使用[35]中的矢量时钟，Levrouw等人的矢量钟表施加的开销对比。基于标量逻辑时钟[22]，介绍了订单重放技术[26]。提交人提供了解的理解和实证评估，支持他们的技术，他们的技术在Time-Netzer的最佳重播和Leblanc'Sinstant重播的两个主要竞争对手方面都在录制开销和内存足迹方面。但是，eval-uation工作负载不会类似于HPC工作负载（例如，排序），并且在最大的offour线程上运行。在ronsse和de bosschere的重复播放中有效的开发期间持续增长次数。这项工作采用多种逻辑时钟：标量时钟在录制期间订购同步性，而窥探矩阵时钟[4]在重放和禁止正确重放时，在重播和播放期间进行竞争检测。重新播放被评估对抗飞溅-2Benchmark套件，在整个稿件的剩余部分中展示25.9％的录制期间最坏情况的执行时间开销。

共享内存应用程序的最新R&R方法解决了广泛的可重现性挑战。在本节中，除了传统的以调试为中心的技术外，我们还将讨论以更轻松的可重现性概念为目标的技术。通过这些技术，我们观察到两个趋势:有针对性的编程模型的多样性增加，以及对硬件辅助方法的兴趣增加。

由于与高保真重播相关的开销，旨在瞄准多核系统的最近技术已经选择放松完全重放录制的运行的要求。相反，这些技术尝试重放运行输出匹配记录的运行输出，而不管产生输出的内部状态转换。 Altekar引入了输出确定性重放（ODR）[2]，它仅记录了FORAMANTEE可重复输出的必要信息的子集，但在重放期间使用搜索过程迭代到再现所录制的RUN的最终状态的线程调度，增加此子集。虽然可以应用于调试的策略，但也可以明确辅助在需要对数值输出的再生率为ITY的情况下。 odr是针对Splash-2基准的套件套件评估的，并展示了1.6倍的平均录制开销.Park引入了一种类似的技术，称为概率重播，用exe-cution素描（pres）[39]。 Pres还放松了普通的重播忠诚要求，这是不同的方式。 Pren而不是更加松散地定义等效的执行，而不是在多次重放运行中逐步接近精确重放。通过构建执行Scketwuring录制，然后迭代地探索与草图的atagree的受约束可能的执行集合，Pren能够在再现错误运行的执行中磨练。在最多四个线程上的Splash-2基准的子集上进行了门位，对于N-Band工作负载，观察到10.6％的尺度执行时间开销。通过容错，Lee介绍了Respec [24]，其中也是目标输出水平的决定论。与此时的最流行的技术相比，在此时间点使用的，将其与受监控的执行同时“重播”，周期性地检查两次运行之间的潜水员，并维护执行执行的最后一致抖动的检查点。 RESPEC仅通过日志记录最常见的同步operations的子集来管理开销，并且从容错协议中借用攻击措施，以纠正自身，当重放的缓解太远时，从监视的执行太远。在Parsec和Splash-2基准套件上，在Parsec和Splash-2基准套件上进行评估Respec。该十年的开始，通过在耦合与分布式存储模型的每个节点上使用共享存储模型，许多代码在HPC系统上并行化实习ode通信（即MPI / X模型）。 Budanur等人，介绍了一个R＆R Techniquefor，在这种情况下研究了Memory效率低下的内存效率低下[6]。 ScalameMtraceReverages扩展了电源常规部分描述符（EPSRD）以跨多个级别的内存层次结构挖掘重复行为，以实现内存跟踪的压缩表示。 Scaramemtrace是针对HPC为中心的工作负载评估的，即SemenoIa AMG基准和两个计算Ker-Nels的TheMPI / OpenMP实现：矩阵 - 矩阵乘法和矢量添加。

在内核的弱比较研究中，Scalamemtrace的压缩在最多64个线程上得到了近恒定的痕量大小，相比之下使用相同迹线的未压缩版本，其与ofthreads的数字呈现。在对AMG基准测试的强大研究中，压缩迹线的线性线性地呈线性，但尺寸减小了50％。ScalameMtrace的重播忠诚有限，即AMG基准WAS仅重放91％的准确性。

虽然第一种重复性为中心的R＆R技术出现并演变，但如前一节，以前部分，以上设计的调试技术仍然是高优先级。用于调试的大拍卖共享内存呈现出必须追溯的事件（即，SharedMemory Accesses）的纯粹次数，以确保所需的高保真重播。 Contem-Porary Debugging的技术通过两种策略解决了这一挑战：首先，在软件级工作第二，通过LeveragingspeCialized硬件来减少录制开销（例如，通过使用带有更紧密的硬件时钟的硬件时钟）。软件级别，我们遵守一个供应商驱动的努力，以满足高度一般的技术的挑战。英特尔建议的PINPLAY [40]，这是一个R＆R技术在其引脚动态仪表框架上。 PINPLAY提供高度灵活的R＆ROPTIONS（例如，子组重播），并提供与其他基于PIN的工具（因此，PUR-SUING PERINGE）的可兼能性。 PINPALL的评估是在现实的HPC工作负载上进行的，包括并行海洋计划，米尔量子色动力学代码和天气预报应用。在这些工作负载中，PINPLAY观察在录制期间36×和147×执行超大的主机，尽管高度高，但表明了对科学应用的高保真重播的录音信息的挑战。从学术界，努力包括重播的重新制作作为满足性问题，并且专业化对任务平行的runtimes.liu等人。的灯[28]介绍了利用满足性水平理论（SMT）求解器的新型软件级技术。光正式地表征了所需的重播水平所需的最小跟踪数据。浅色侧重于日志记录共享内存访问的流量依赖性，然后在重放期间生成一个针对记录的流依赖性的线程计划，该线程计划将其作为可满足问题构成其作为可满足问题。在从科学应用程序到Web服务器和Web爬网应用程序中评估光线的各种基准标记。

虽然本节中的所有技术都集中在录制线程的行为上，但在基于任务的并行运行时（如Cilk Plus）中，这不是一个选项，因为Runtime'sscheduler可以使用从运行中运行的可变数量的线程来运行Fashion.在这种情况下需要所谓的“处理器忘记”R＆R. Porride [51]是The Pilk的第一个如此，它针对Cilk Plus程序，其中共享对象是锁定的。 Porride维护每锁记录，而不是维护每个线程记录，通过使用表示在录制期间所观察到的关系之前的额外边缘来控制Cilk Scheduler来符合这些记录的访问订单。粥是针对不同集合的套生的，包括来自Parsec套件的两个，以及三个非etministic图算法，基于问题的基准套件。录制分阶段的执行时间开销。 Chapp，K. Sato，D.H. Ahn，M. Taufer2018，Vol。 5，117号

从基本上可以忽略不计3.39倍，所有基准测试的平均值为1.62倍。 TheGraph基准测试显示出录制和重放的最高开销。与R＆R的新软件级别方法平行，研究人员致力于在硬件上努力，并提出未来的硬件增强。在维护调试Howeret al所需的重播保真程度的同时努力减振开销。介绍了Rerun [17]，其建议使用每种经合组织的相对较少的专用内存，与标量逻辑时钟耦合以建立部分orderOn原子发作，即一个线程不与任何其他线路进行比赛。在2010年代中提出了额外的硬件辅助技术，最值得注意的是QuickRec [41]，其提出了Intel架构的扩展，以支持有效的记录和重播，并利用硬件辅助虚拟化扩展。 QuickRec在最多八个线程上的Splash-2基准上进行了评估，而Samsara则评估最多四个线程的PARSEC基准。最近，刀具脚轮[30]已经开发出来提供录制，以便提供这种较低的录音默认情况下可以留下，而不是仅在特定调试运行过程中使用.Castor使用依赖于同步硬件时钟的时间戳过程。与其他工具相比之下，Castor监视器重播执行并动态评估是否有足够匹配的记录。 Castor维护非替代的每线程记录，它们定期聚集到主记录。 CASTOR通过在多个级别使用事务存储器，通过组合LLVM通过，库互操作和线程测井，在录制和重放期间录制和重放过程中的主要贡献是日志吞吐量的主要贡献。蓖麻采用PARSEC基准套件评估。虽然Castor展示了大多数PARSEC基准测试的低录制开销，但在10个线程上运行时，RA-渗透率基准测试率高高达25％。作者声称这是由于放热对日志聚合对高速缓存的影响的敏感性，因此周期性地不堪重负的聚合器。

3.分布式内存的记录和重放

在本节中，我们讨论了目标分布式内存模型的R＆R技术。作为Thronessage-Crosse界面（MPI）[32]成为分布式内存HPCapplications的TheDefactandard，R＆R技术越来越多地定位MPI应用。因此，我们在本节中重点关注技术。首先，我们讨论R＆R技术的早期方法，以追求可扩展性的方式转移数据重放。接下来，我们确定逻辑时钟算法的集成到逻辑时钟算法的集成时重新展现的进步时代。最后，我们识别我们分为三个主要研究方向的信息技术：一是导向的调试技术，重点是高保真重播;其次，技术面向通信特征和性能分析的再制剂。第三，Techniquesthat跨越R＆R和容错之间的线路。

用于分布式内存应用程序的R＆R技术的最早时代将数据重新显示了三个原因。首先，早期的R＆R工具是在调试社区中开发的，而不是HPC社区，所以记录消息的内容以牺牲可扩展性为代价提供丰富的调试信息，被视为可接受的权衡。其次，HPC系统的消息传递和重播技术：SULECH18普通的前沿和创新

MPI等标准仍然成熟，这限制了订单重播技术的发展。第三，数据重放不需要重播所有进程，允许用户重点授予错误的进程。来自此时代的代表性数据重放技术包括Bugnet [11]和Recap [38]。 Bugnet记录了所有消息内容，并且代表了三个工具三个设计，以减轻在调试并行代码时减轻非eterminism的负担。重复以两种方式表示前一步。首先，最突出地，RECAP是第一个提供子组重播的工具;识别在调试上下文中，需要重播才能重播进程。其次，RECAP明确承认，速度生成的速率限制了技术的适用性.HPC社区开始认识到R＆R工具的需求，顺序重播作为数据重放的有吸引力的替代品。勒布朗的即时重播[23]是最早的技术，如第2.1节所述。即时重播模型所有进程间通信都访问共享对象，并在repowsphase期间为每个此类访问分配版本号。这种方法避免了复制和存储消息缓冲区内容的需要，以及对调试方案的SOIS具有吸引力，其中应用程序可能需要长时间的错误清单。即时重播的实证评估显示了可以通过订单重播方法实现的太空​​飞船。具体而言，LEBLANC显示，对于64个进程的Agaussian消除工作负载，即时重放的迹线占用300x的小于数据重放迹线，其中记录了所有消息的内容。即时重播呈现出挑战的追踪追踪追踪的所有消息的内容。被黎卢等人识别和解决。在他们的提议重播[25]中。 leu观察到即时重播的版本控制算法需要额外的消息，这可能会干扰调试。执行重放约束自身发送不发送其他消息。此外，Leu采取了阶段，迈出了相当于执行的概念（即，这意味着说重播是“正确”？）。最后，Leu明确地解决了拦截和非阻塞通信的重播 - 在现有工作中的区分。在整个初步的发展期间，我们观察到将研究的关键驾驶问题引导至今的关键驾驶问题（即，媒体乘坐的折衷和执行时间开销之间的权衡）。此外，系统建筑的演变是管理上述权衡的必要性。这导致在整个20世纪90年代和2000年代早期讨论的研究生R＆R技术突发。

DebuggingOrder-replay 技术是为了响应 HPC 对可扩展 R&R 日益增长的需求而开发的。 在 1990 年代和 2000 年代初期，出现了两个相关的设计目标。 首先，最小化内存开销和跟踪大小被认为是将 R&R 应用于长时间运行的、需要大量内存的 HPC 应用程序所必需的。 其次，最小化执行时间开销被认为是必要的，因为即使事件时间的微小变化也可以防止在记录阶段出现错误。 逻辑时钟 [22]（即，用于跨多个进程建立事件顺序的算法）成为解决这两个问题的工具。 这两个挑战将在以下部分中讨论，讨论与逻辑时钟和重放技术混合相关的增强功能。 D． Chapp, K. Sato, D.H. Ahn, M. Taufer2018, Vol. 5、119号

3.2.1。 使用逻辑时钟增强顺序重放技术Netzer 在 Optimal Replay [36] 方面的工作是 1990 年代 R&R 研究方向的典型。 总体思路是使用向量值逻辑时钟 [12] 在运行时检测潜在的消息竞争，并仅记录足够的信息以使重放期间的竞争结果与记录运行期间的结果相匹配。 可以安全地假设所有其他非竞赛通信在没有记录跟踪的指导下重复。 在一组计算内核（例如矩阵乘法）上对最优重放的经验评估表明，只有 1-14% 的消息需要跟踪，从而显着降低了跟踪开销。 除了 Optimal Replay，Netzer 还引入了 Incremental Replay [34]，它将检查点与 R&R 中常见的消息记录相结合，允许重播执行“快进”到实际发生错误的执行区域。 虽然对调试的关注很明确，但这项工作表明了调试和 HPC 容错社区之间的概念重叠——这种趋势一直持续到现在。在 Netzer 关于 Optimal Replay 的工作之后不久，MPI 被标准化并被 HPC 社区接受为 分布式内存科学应用的事实机制。因此，分布式内存应用的R&R技术开始专门针对MPI。 最早的例子是 Clemencon 在 Annai 编程环境的并行调试工具上的工作 [9]。 Clemencon 的技术建立在 Optimal Replay 的基于矢量时钟的竞赛检测器之上，为每个进程添加了一个额外的标量逻辑时钟。 这项工作的意义不仅在于它对逻辑时钟的使用，而且在于它解决了非阻塞通信中固有的不确定性，包括非阻塞探测。 这项工作在通信微基准测试以及更多面向 HPC 的基准测试（例如 BiCGSTAB）上进行了评估，最多可支持 64 个进程。尽管矢量时钟具有确定何时必须记录事件的能力，但所有消息上的背负矢量时间戳的需求变为 随着系统规模的增加，越来越令人望而却步。 ROLTMP[46] 试图通过在记录期间使用标量逻辑时钟来确定何时必须显式记录事件来克服这种可伸缩性障碍，并在重放期间加上额外的检查以补偿标量时钟跨进程排序事件的有限能力 [8]。正如 MPI 所见 在 HPC 中越来越广泛地采用，R&R 技术出现了，它利用 MPI 的点对点通信语义来减少具有有限不确定性的应用程序的记录开销。 Kranzlmuller 在非确定性程序评估器 (NOPE) [20] 中引入了这种技术。 这项工作的新颖之处不仅在于它放弃了逻辑时钟，而是采用了一种更简单的日志机制，该机制利用了 MPI 的非超车规则，而且 NOPE 支持重播替代执行（即，一个执行被记录，但多个可能的执行，包括记录的 一，可能会重播）。 NOPE 的缺点是它假设接收方不确定性的唯一来源是通配符接收，并且记录的应用程序是发送确定性的 [7]。除了利用 MPI 本身的特性之外，R&R 工具是为了响应 HPC 开发人员在他们的代码中写入的特定编程习惯。 De Kergommeaux 等人。 开发了 MPL∗[18]，它针对在轮询循环中使用非阻塞测试功能的 MPI 应用程序的正确重放。 这种编程习惯通常用于重叠通信和计算，但一个结果是测试的数量变得不确定。MPL\*紧凑地表示测试调用的序列，以便使用测试调用数量的应用程序（例如，用于控制流） 可以正确回放。 Kranzlmuller 后来迈出了一步 用于 HPC 系统的记录和回放技术：调查20 超级计算前沿和创新

除此之外，选择将 R&R 功能集成到 MPI 库本身 [19]。 这种方案的主要优点是在调试环境中为用户提供方便，因为无需检测代码、与 MPI 之外的其他库链接或管理跟踪。 Bouteiller 等人的后期工作。 产生了 Retrospect [5]，它将 R&R 直接集成到 MPI 实现的较低级别的通信例程中，并且可以根据用户需求在数据重放和订单重放策略之间切换。 Retrospect 为分布式内存 R&R 技术的经验评估设立了新标准，因为它在 NAS Parallel Benchmarkson 上进行了多达 1,024 个进程的评估。

R＆R的当代研究除了去颠簸之外还沿两种新颖的方向推进：再现性和容错。首先，出现了针对再现性的莫雷祥的定义的一类R＆R技术（例如，用于表征申请或学习性能可移植性的沟通行为）。这些技术不会通过调试为导向的技术的细粒度，高保真重放，而是对更大的执行进行优先考虑。其次，调试和故障 - 托利通信之间的交叉授粉导致了一类容错协议，涉及消息记录传统R＆R的杠杆龄技术。首先，我们讨论这些新的方向，我们通过讨论以与以中心为中心的记录和重播的当代方法进行结论。

虽然调试仍然是R＆R的主要焦点，但近期惠普社区的兴趣来自重现性（例如，绩效，沟通行为，模拟的科学结果）具有激励R＆R技术，这些技术与更轻松的保险费相比调试案例.A这个趋势的显着例子是Scalatrace工具。原始Scalatrace [37]工具提出了跟踪压缩技术，其为MPID提供了几乎恒定的迹线。 Chapp，K. Sato，D.H. Ahn，M. Taufer2018，Vol。 5，121号

具有足够常规通信的应用。常规部分描述符（RSD）和递归递归对应力电源RSD用于紧凑地代表MPI CommunicationEvents，它与一套MPI特定的编码技术相结合，显着降低了具有相对较小的非术语的应用程序的尺寸。alatrace的直接继承者，Scala-H -trace [53]提供概率重播。虽然坚定地复制了细粒度的MPI事件，例如单独的消息接收，Scala-H轨迹允许用户指定调整参数，该调整参数控制何种展开的保真度与录制可扩展性之间的权衡。通过上述前身的无损跟踪方法，Scala-H-Trace能够在最多2,048个进程上记录并行海洋程序的执行时实现近常数跟踪文件大小。相比之下，原始的Scalatrace是能够在最多1,1024个进程上记录相同的执行。此外，Scala-H-TraceMementStrated恒定跟踪尺寸在2,048个过程中的NPB共轭梯度工作负载。重要的警告是使用Scala-H轨迹的最低重播保真设置获得这些结果，以及用于评估重播正确性的度量是整体执行时间。尽管对调试的适用性有限，但为科学重复性提供了概率重播的技术，并且为了确保申请的沟通行为具有与程序员意图的特征.Scala-H-Trace的继任者，Scalatrace II [52]，也包含概率的重播。 Proposatas在Scalatrace中开发的压缩技术进行了更加不规则行为的重大重新设计，Scalatrace II在NPB，Sweep3Dneutron传输内核和最多400个过程上的并行海洋计划中评估了Scalatrace II。与Scala-H-Trace一样，Scalatrace II执行MPI事件的有损压缩，因此其正确性标准是重播是总体执行时间。在所有评估工作负载中，Scalatrace II在重放时间和最初观察到的执行时间之间实现了5.7％的普通视为时间误差之间的真实性。与其前身一样，概率重播Scalatrace II提供用于实现通信行为的粗粒度重复性。在各种尺寸的模板，NPB的子集和猛禽，纵向框架上支持支持本评估中使用的自适应网格细化的急剧性致力于的粗大粒度的重复性。运行Affrodynamics模拟。这些工作负载在最多512个进程上记录，并且在MOSTCASES中在录制期间展示了几乎恒定的跟踪尺寸和内存占用。然而，来自NPB的共轭梯度和傅立叶变换基准产生的跟踪尺寸随着处理器的数量而产生的。在共轭梯度基准测试的情况下，在增加的跟踪大小中的异步点对点通信的情况被确认。最近，Zhai等人。 [57]引入了“代表重放”，其针对穿孔预测。代表性重放允许用户在有限平台（例如，群集的单个节点）上记录并行APPLICED的执行，并通过受控的重放执行在整个平台上获取对相同应用程序的直观时间的预测。

参考文献

1. Hennessy J L, Patterson D A. A new golden age for computer architecture[J]. Communications of the ACM, 2019, 62(2): 48-60.
2. Celio C, Chiu PF, Asanović K, et al. Broom: an open-source out-of-order processor with resilient low-voltage operation in 28-nm CMOS [J]. IEEE Micro, 2019, 39(2):52-60.
3. O'Callahan R, Jones C, Froyd N, et al. Engineering record and replay for deployability[C]. 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017: 377-389.
4. Narayanasamy S, Pokam G, Calder B. Bugnet: Continuously recording program execution for deterministic replay debugging[C]. 32nd International Symposium on Computer Architecture (ISCA'05), IEEE, 2005: 284-295.
5. Srinivasan S M, Kandula S, Andrews C R, et al. Flashback: A lightweight extension for rollback and deterministic replay for software debugging[C]. USENIX Annual Technical Conference, General Track, 2004: 29-44.
6. Saito Y. Jockey: a user-space library for record-replay debugging[C]. Proceedings of the sixth international symposium on Automated analysis-driven debugging, 2005, 69-76.
7. Dunlap G W, King S T, Cinar S, et al. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay[C]. 5th Symposium on Operating Systems Design and Implementation (OSDI 02). 2002.
8. Sheldon M, Weissman G V B. Retrace: Collecting execution trace with virtual machine deterministic replay[C]. Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2007). 2007.
9. Navarro Leija O S, Shiptoski K, Scott R G, et al. Reproducible containers[C]. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2020: 167-182.
10. Aviram A, Weng S C, Hu S, et al. Efficient System-Enforced Deterministic Parallelism[C]. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
11. Hunt N, Bergan T, Ceze L, et al. DDOS: taming nondeterminism in distributed systems[J]. ACM SIGPLAN Notices, 2013, 48(4): 499-508.
12. Bergan T, Hunt N, Ceze L, et al. Deterministic Process Groups in dOS[C]. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10), 2010.
13. Devecsery D, Chow M, Dou X, et al. Eidetic systems[C]. 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 525-540.
14. A. Waterman and Y. Lee. Spike - RISC-V ISA Simulator[OL]. https://github.com/riscv/ riscv-isa-sim.
15. Bellard F. QEMU, a fast and portable dynamic translator[C]. USENIX annual technical conference, FREENIX Track. 2005, 41(46): 10.5555.
16. Karandikar S, Biancolin D, Amid A, et al. Using FireSim to Enable Agile End-to-End RISC-V Computer Architecture Research[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2019.
17. Clark M, Hoult B. rv8: a high performance RISC-V to x86 binary translator[C]. First Workshop on Computer Architecture Research with RISC-V (CARRV). 2017.
18. Asanovic K, Avizienis R, Bachrach J, et al. The rocket chip generator[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, 4.
19. Asanovic K, Patterson D A, Celio C. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor[R]. University of California at Berkeley Berkeley United States, 2015.
20. Balkind J, Lim K, Gao F, et al. OpenPiton+ Ariane: The first open-source, SMP Linux-booting RISC-V system scaling from one to many cores[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2019: 1-6.
21. Vega L, Taylor M B. RV-IOV: Tethering RISC-V Processors via Scalable I/O Virtualization[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2017.
22. Roelke A, Stan M R. Risc5: Implementing the RISC-V ISA in gem5[C]. First Workshop on Computer Architecture Research with RISC-V (CARRV). 2017, 7(17).
23. Ta T, Cheng L, Batten C. Simulating multi-core RISC-V systems in gem5[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2018.
24. Hin P Y H, Liao X, Cui J, et al. Supporting RISC-V Full System Simulation in gem5[C]. Workshop on Computer Architecture Research with RISC-V (CARRV). 2021.
25. Butko A, Garibotti R, Ost L, et al. Accuracy evaluation of gem5 simulator system[C]. 7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC). IEEE, 2012: 1-7.