计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140105
52(1):45-55,2015

一种用于多线程程序性能分析的重放系统

郑 龙 廖小飞 吴 松 金 海
(服务计算技术与系统教育部重点实验室(华中科技大学) 武汉 430074)
(华中科技大学计算机科学与技术学院 武汉 430074)
(longzh@hust.edu.cn)

A Replay System for Performance Analysis of Multi-Threaded Programs

Zheng Long, Liao Xiaofei, Wu Song, and Jin Hai
(*Key Laboratory of Services Computing Technology and System (Huazhong University of Science and Technology), Ministry of Education, Wuhan* 430074)
(*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract In recent years, it is a hotspot for program analysis to detect performance bugs in multi-threaded applications. However, traditional record/replay systems focusing on concurrent anomalies have many limitations to tackle the issues of performance bugs, such as replay overhead and imprecision of replay-based execution time. To cope with the problems above, this paper proposes an improved replay system PerfPlay which can be used for the performance analysis of multi-threaded programs. To be specific, we first collect and analyze the requisite information for the program performance. Secondly, the different replay strategies are discussed and then we present a novel schedule-driven strategy to ensure the performance fidelity of replay system. Finally, we study the classical performance problem of "inter-thread unnecessary lock contention" under the framework of PerfPlay. Compared with the traditional replay strategies, our experimental results demonstrate the performance fidelity of PerfPlay. Through the case study, we find a few performance bugs in real-world and further verify the effectiveness of PerfPlay.

Key words performance bug; replay; case study; multi-threaded, unnecessary lock contention

摘 要 近年来,多线程程序中性能 bug 问题越来越突出.传统用于检测并发错误的记录/重放系统存在重放开销和执行时间不精确等问题,因此不适于对性能 bug 的研究.针对上述问题,提出了一种可用于多线程程序性能分析的重放系统——PerfPlay.首先,分析了用于程序性能分析时必要的程序信息;其次,基于程序执行轨迹,探讨了不同的重放策略,并提出了基于程序调度的重放策略,以保证重放系统的性能保真度;最后,基于提出的性能重放系统,进一步研究了经典的"线程间不必要锁竞争"所造成的性能问题.通过与传统的重放策略作比较,PerfPlay 保证了重放系统的性能保证度.并经过案例研究,发现并进一步验证了若干真实的多线程程序性能问题.

关键词 性能 bug;重放;案例研究;多线程;不必要锁竞争

中图法分类号 TP311

收稿日期:2014-10-13;**修回日期**:2014-11-11

基金项目:国家自然科学基金项目(61272408,61322210);高等学校博士学科点专项科研基金项目(20130142110048);国家"八六三"高技术研究发展计划基金项目(2012AA010905)

通信作者:廖小飞(xfliao@hust.edu.cn)

程序分析技术是一类以程序为处理对象,按照特定的需求(如程序正确性和资源优化等)对程序行为进行分析的方法,它在程序理解、程序测试、程序优化和程序重构等方面有着广泛的应用[1].

在早期研究中,大量基于追踪驱动(trace-driven)的重放分析技术[2-7]用于调试多线程程序并发错误,如数据竞争、死锁及原子性违反等.为了重现并发错误,传统重放系统主要工作流程为:1)预先记录程序执行轨迹;2)按照特定的重放策略执行已记录的程序轨迹;3)根据重放执行结果确认并重现程序中的并发错误.随着多线程编程语言规范的发展与成熟[8-9],多线程程序中性能 bug 问题愈来愈突出[10].相对于并发错误而言,性能 bug 是一类仅导致程序性能下降或资源浪费的编程问题,它并不影响程序执行的正确性.但是,上述传统重放系统流程2)所采用的重放执行策略通常会不同程度地导致重放执行轨迹不符合真实的程序执行情况.因此,传统重放系统以强制程序执行来检测并发错误的方式不再适用于对多线程程序性能问题的研究.例如,CoreDet[11]以确定的时序调度所有的共享访存行为(load/store 指令),这保证了程序执行的确定性,但却造成相较于原始执行约 1.2～6 倍的执行时间.Kendo[12]仅保证同步点(如 lock/unlock 锁操作)的确定性时序,因此提供了高效的确定性执行技术,但是仍然造成 16%的性能下降.也正是这部分性能开销严重地影响了传统重放系统程序性能分析时的准确性.

近些年也有不少工作致力于程序性能分析技术的研究,例如调用堆栈 profiling 技术[13]、等待图分析[14]等.但这些工作均需要耗费大量的 profiling 开销或人工地去定位性能问题.更重要的是它们只是用于测量相关软件组件的执行时间,因此不能解决"多线程程序中哪些组件造成性能浪费"、"如果该组件被优化将会带来多少性能收益"等研究问题.

本文提出了一种改进的可用于多线程程序性能分析的重放系统 PerfPlay.为了克服传统重放系统中执行性能不精确的问题,我们对传统重放系统的工作流程进行了详细地分析:

1) 分析了程序性能重放所必需的记录信息;同时提出了一种高效的执行性能等价的程序轨迹记录策略以减少记录开销;

2) 分析了重放阶段中造成程序性能不稳定的关键因素,并基于此提出了一种新颖的性能重放策略,以保证重放执行的性能保真度;

3) 分析了基于重放技术的程序执行的性能分布,并进一步探讨了基于重放的程序执行和真实的程序执行之间的关系.

基于上述性能重放分析,多线程程序中相关性能研究的工作流程转化为:1)预先将程序执行轨迹记录到记录文件中;2)将需要研究的性能问题在记录文件中消除;3)按照提出的性能重放策略分别对原始记录文件和修改后的记录文件进行重放;4)通过对比分析两次重放执行的结果,进而理解和定量评估所研究的性能问题的重要性及意义.

为了验证 PerfPlay 性能重放系统的有效性,本文对经典的"线程间不必要锁竞争"所造成的性能影响问题① 进行了案例研究.探究在不借助硬件支撑的条件下,如何用性能重放系统 PerfPlay 去理解和定量分析该问题.结果表明,基于 PerfPlay 的重放执行可以约束真实的程序执行,并严格保证了程序执行的性能保证度.本文亦通过逆向研究的方法定量研究了若干由 PerfPlay 框架发现的不必要锁竞争问题,测试数据验证了 PerfPlay 在性能分析方面的有效性.

1 传统确定性重放技术

研究人员提出了不同的确定性执行技术来消除多线程程序执行的不确定性,进而使得并发错误易于检测.但是由于技术本身的缺陷,这些技术在不同程度上强制延长了程序执行的时间.对于同一输入条件下的多个调度,图 1 描述了不同确定性执行系统的技术特征.BugNet[17]和 FDR[18]按照等量的执行指令数目交替串行化地执行各线程(即将并行程序当串行程序来执行),因此带来大量的时间开销,并不适于并行重放.通过严格地确定共享行为(包括共享访存和同步事件等)的执行时序,PinPlay[6]和 CoreDet[11]部分允许了本地线程事件的并行重放,但是仍然造成相较于原始执行约 1.2～6 倍的执行时间,特别是对内存密集型应用,性能下降更为严重.为了提高确定性执行技术的效率,Kendo[12]进一步允许共享访存事件的并行执行,仅保证同步点(如

① 线程间不必要锁竞争性能影响问题首次由 Rajwar 和 Goodman 提出与初步解决[15],该文的改进工作[16]被计算机体系结构领域顶级国际学术会议 ASPLOS 评为最有影响力论文.

lock/unlock,signal/wait,barrier 等)的确定性时序,但是仍然造成约 16%的性能影响,且 Kendo 的执行策略不适用于含有数据竞争的程序.如图 1 所示,针对不同的线程调度(如 Schedule1 和 Schedule2),CoreDet 和 Kendo 均造成了不同程度的性能损失,这种执行的不精确性也使得传统的确定性重放技术不再适用于对多线程程序中性能问题的研究.为了消除因为传统确定性技术所导致的程序性能开销问题,本文提出了一种新颖的性能重放策略,以保证重放执行的性能保真度.基于此策略,本文进而提出了一种可用于多线程程序性能分析的改进型重放系统 PerfPlay,关于 PerfPlay 性能重放系统的详细讨论将在第 2 节展开.

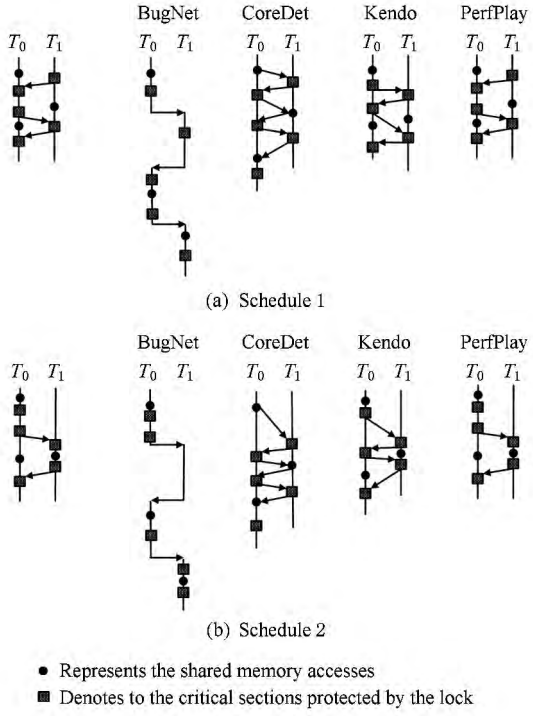


Fig. 1　The technical characteristic of different deterministic execution techniques.

图 1　不同确定性执行技术的执行特征

2　PerfPlay 性能重放系统

本节从性能信息记录、性能重放执行、基于性能重放的性能研究 3 个层面对 PerfPlay 进行详细的探讨.

2.1　性能重放信息记录及记录方式

1) 全域记录.全域记录是一种最直接的准确重放程序性能的方法.它完备地记录程序所有执行信息,包括寄存器状态、程序状态、局部和共享内存状态、完整的执行指令信息(load/store)等.显然,这种记录方式将会带来高额的记录开销,例如,Zhou 等人[19]和 Lee 等人[20]在记录完整的程序执行信息时记录文件以 2 MBps～200 MBps 的速度爆炸式增长.

2) 选择性记录.为了克服全域记录高开销的特征,PerfPlay 采用了选择性的记录方式.对不关心的代码区域或特定的代码区域(比如系统调用、库函数及自旋锁等),该方式记录了代码区域执行前和执行后的程序状态和值,并维护了程序运行时全局影子内存;在重放该段代码时,通过恢复该段代码程序处理器、寄存器的状态与值和动态维护影子内存信息来模拟该段代码的执行场景.选择性记录的方式减少了大量的日志记录开销,也使得重放系统的运行时开销变得更小.

图 2 描述了选择性记录方式的某些详细技术特征,展示了一段简化的代码示例:其先调用系统调用 sys_call_1,然后进行用户操作 do_sth,进而再次调用系统调用 sys_call_2.若所研究性能的问题在 do_sth 中,选择性记录主要步骤如下:

1) 在记录阶段,对特定代码区域进行插桩,记录它们的运行时间和运行后的系统状态.例如,图 2 中,系统调用 sys_call_1 某次运行的系统时间为 t_1,其运行后的系统状态为 $sys\text{-}call_1\text{-}state$.

2) 在用上述记录信息重放程序时,特定代码区域将会用 sleep 和 restore 进行等价执行.例如,图 2 中,系统调用 sys_call_1 的真实执行会被 $sleep(t_1)$和 $restore(sys\text{-}call_1\text{-}state)$等价替代.

如图 2 所示,在记录阶段,选择性记录不用记录 sys_call_1 和 sys_call_2 完整的运行时信息,只需记录

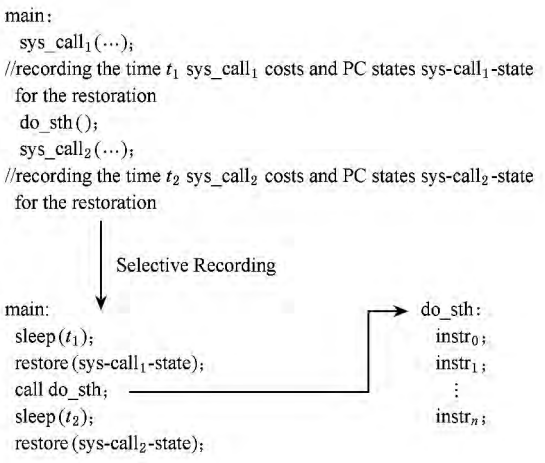


Fig. 2　The technical features of selective recording technique.

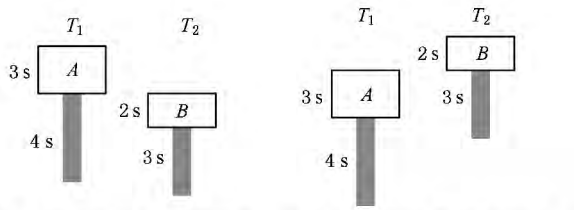
图 2　选择性记录方式的技术特征

其运行时间和运行后的系统状态即可. 因此,相较于全域记录,选择性记录节省了大量的记录开销.

2.2 性能保真度

2.2.1 性能波动的影响因素

对于一个已知的记录文件,将要重放的程序调度有如下可以确定的恒定信息:1)程序启动的线程数;2)每个线程的执行路径;3)各执行路径上的指令. 因此,若多次并行重放该记录文件,程序可能的执行调度将很大程度上取决于线程间的锁交错,这也进而造成了性能上的波动(即性能不稳定性). 如图 3 所示,临界区 A 和临界区 B 共同竞争同一锁资源,若 A 优先 B 执行,程序执行时间为 8 s;若 B 先于 A 执行,程序执行耗时 9 s;即锁竞争的调度顺序造成了程序性能上的不稳定. 为了提高性能分析的精确性,我们需要保证程序重放执行的稳定性(见 2.2.2 节).



(a) If A precedes B, the program costs 8 s (b) If B precedes A, the program costs 9 s

Fig. 3 The performance fluctuation arising from two possible lock interleavings.

图 3 线程间锁交错所导致的性能波动

2.2.2 基于程序调度的弱化锁时序约束策略

为了保证多线程程序执行性能的稳定性,我们提出了弱化的锁时序约束条件(relaxed locking serialization constraint, RLSC),即基于程序运行时的锁调度时序,保证每次重放执行时的锁时序与之一致. 以图 3 为例,若在记录某次多线程程序执行轨迹到记录文件时,该程序的调度时序为 $A\rightarrow B$,那么 RLSC 同时将记录下该锁调度时序到记录文件中. 在以后每次并行重放该日志文件时,程序的锁时序将会永远是 $A\rightarrow B$. RLSC 严格遵守程序执行的真实调度,同时也不会因为线程调度因素造成性能分析的不稳定性.

Kendo 也采用了锁时序的策略确定程序的执行,但该策略是基于程序输入集的,因此在不同程度上造成了程序性能的不确定,具体来说,如图 1 所示,若某次程序的调度时序为 Schedule1,在同样输入条件下 Kendo 将会交替不同线程中的锁获取时序. 如在 Schedule1 的情况下,尽管线程 2 的第 1 个临界区操作先于线程 1 中的第 1 个临界区操作执行,Kendo 仍会把线程 2 中第 1 个临界区操作推迟到线程 1 中第 1 个临界区操作结束. 这种确定性执行方式在并发错误重现研究方面有着很大的应用前景,但是在性能分析方面有着诸多局限性:1)Kendo 可能会强制执行一个不真实的线程调度;2)强制的程序调度延长了程序执行,带来了一定的性能开销,这也使得多线程程序的性能分析变得不精确. 相较于 Kendo 基于输入集的锁时序策略(在同一输入条件下,如 Schedule1 和 Schedule2,程序总会执行同一程序调度),RLSC 是一种基于程序调度的锁时序策略,不同的调度都会有不同的锁时序. 因此 RLSC 严格保证了程序调度的准确性和重放执行的性能稳定性(性能保真度).

性质 1. 对于同一程序执行轨迹记录文件,RLSC 保证了并行重放执行的程序性能保真度.

证明. 考虑这样一个典型的锁同步模型:假如一个程序含有两个线程,两个线程的执行顺序分别是 $\{SG_1, A, SG_2\}$ 和 $\{SG'_1, B, SG'_2\}$. 其中 A 和 B 是由同一锁保护的临界区资源,SG 表示某一个程序段. 在程序执行轨迹记录阶段,假如临界区资源 A 和 B 的时序关系为 $A\rightarrow B$,并且各程序段和临界区 $\{SG_1, A, SG_2, SG'_1, B, SG'_2\}$ 执行到其最后一个指令时所耗费的时间分别为 $\{t_1, t_A, t_2, t'_1, t_B, t'_2\}$. 对于后续的重放,有 3 种可能的调度情况:

1) 若 $t_1 > t'_1$,那么临界区 B 将会优于临界区 A 执行,但是该执行调度违反了所观察程序真实的执行情况;

2) 若 $t_1 < t'_1$,那么临界区 A 将会优于临界区 B 执行,这符合所观察程序真实的执行情况;

3) 若 $t_1 = t'_1$,临界区 A 和临界区 B 共同公平地竞争锁资源,最终谁优先执行取决于系统调用. 在这种情况下①,RLSC 强制锁时序为记录阶段的锁调度时序,即 $A\rightarrow B$;因此,保证了同样的原始执行.

一个已知的多线程程序执行轨迹正是由若干个上述锁同步模型组成的,基于此,RLSC 保证了同一程序执行轨迹记录文件的重放执行的性能保真度.

证毕.

2.3 重放执行时间与真实执行时间之间的关系

目前,研究人员对程序的性能研究基本都是基

① 在大多数执行情况下,多个线程同时公平地(即 $t_1 = t'_1$)竞争同一锁资源在多线程程序中是非常普遍的一种现象.

于程序动态执行模型[21](execution-based model):程序运行的同时进行程序的性能分析.程序动态执行的分析方式有诸多优势:1)一步就能完成对程序性能的分析;2)可以分析程序的不同分支;3)适用于底层 OS 的性能分析.该方法常用于衡量程序不同组件的执行性能(即可以回答"各软件组件耗时多少"的问题),但是无法分析和修改程序性能数据,因此不能回答诸如"程序中哪些组件有性能上的损耗"、"若该软件组件以正确而高效的方式运行,性能上可能会提升多少"等问题.

为了解决上述问题,本文采用了追踪驱动模型(trace-driven model)的方式,并对其进行改进,使之适用于对多线程程序的性能研究.在该模型中,程序首先被插桩,在该插桩程序运行时将相关的程序信息(比如指令操作码、数据地址、程序分支信息等)写入记录文件,最后将记录文件读入模拟器或被直接运行进而定量分析性能问题.相比于程序动态执行模型,追踪驱动模型有着两方面的优势:1)若记录日志文件格式已知,程序性能研究可以在不同的机器上反复观察与重现;2)程序性能信息可以被精确地离线分析并作相应的修改,进而用于性能损耗分析和对比.但是,由于需要将程序记录信息从磁盘读入内存,该模型通常会造成很大的程序重放开销,这可能限制了该模型的性能评估.接下来,我们介绍一种方法来说明追踪驱动模型下性能评估的准确性.

假设某个记录文件的大小为 N 字节.记录文件(数据集)通常会远远大于当前内存可用容量,在读取记录文件信息时通常是分块读取.假设每次读取块大小为 K 字节,因此,程序被分为 $\lceil N/K \rceil$ 块;若每块的读操作耗时为 T_r,程序各块读取后的执行时间分别为 $\{t_0, t_1, \cdots, t_{\lceil N/K \rceil}\}$.于是,我们可以得到追踪驱动模型的重放执行时间 T_{replay},可简化为

$$T_{\text{replay}} = NT_r/K + T_{\text{format}} + (1+\sigma)T_{\text{real}}, \quad (1)$$

其中,T_{format} 为将字符格式的记录文件格式读取后转换为机器可识别的指令格式所需的时间;σ 为模拟器执行框架影响因素(例如,相较于原始执行,PIN 插桩框架下的程序执行延长因素 $\sigma \approx 0 \sim 5\%$[22].),若是直接运行于物理机,$\sigma$ 的值为 0.在实际测试中,因子 N 和 K 是给定的,而 T_{format} 和 T_{real} 可以用内部定时器的方式被量化统计.式(1)表明:如果在因子 N 和 K 已知的情况下,追踪驱动模型的重放执行时间很大程度上取决于程序的真实执行时间,且两者存在一种线性关系.

2.4　基于 PerfPlay 框架的性能研究

解决了上述关键性的技术挑战后,我们对传统的重放系统构架进行了改进,以使之能利用上面提出的重放策略 RLSC 来进行程序性能分析.改进后的重放系统 PerfPlay 的系统构架如图 4 所示:

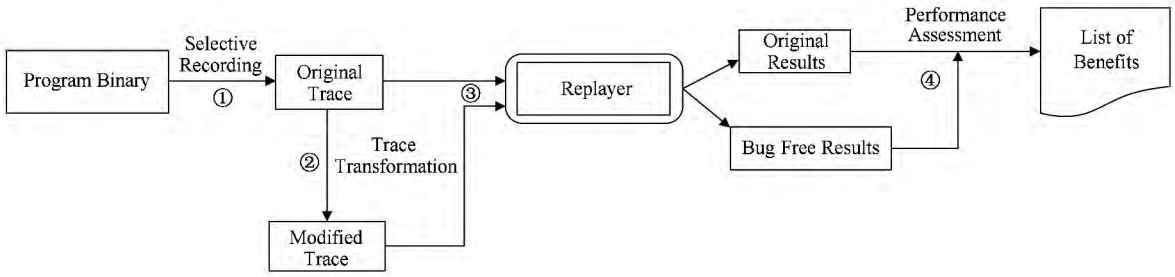


Fig. 4　The architecture of PerfPlay.

图 4　PerfPlay 系统结构

PerfPlay 完全操作于程序二进制层面,无需程序源代码,其工作流程主要分为 4 步:

1) 插桩程序二进制中待评估性能的程序点,采用选择性记录的方式记录插桩后的二进制程序执行,并生成原始程序执行记录文件;

2) 在记录的原始记录文件中,观察待评估性能的程序点的运行时特征(例如锁调度时序、计算资源消耗及程序运行时间戳等信息),进而观测或消除它们对程序执行性能的影响,即保证该部分程序点以正确地方式运行;

3) 利用弱化的锁时序约束策略(RLSC)对原始记录文件和修改后的记录日志文件分别进行重放,生成重放执行性能日志;

4) 对比分析两次重放执行的性能数据,观察和定量评估待研究性能问题的重要性和意义.

上述 4 步中,第 2 步是所有步骤中最重要和最灵活的一环;它需要根据不同的待研究问题的变化而衍生出不同的应对策略.针对不同的待评估性能问题,我们需要观测的运行时特征会有差异."线程间不必要锁竞争问题"(将在第 3 节讨论)的主要运行

时特征是锁调度时序;又如"衡量程序组件的性能"的主要是运行时特征是程序运行时间戳信息.这里的运行时特征需要分析人员针对不同的性能问题去具体化.例如,如果我们需要统计某个函数的运行时间,我们可以在记录文件中对其时间戳进行标识,统计其运行时间区间.

为了验证 PerfPlay 系统的有效性,借助于经典的"线程间不必要锁竞争"问题,我们基于 PerfPlay 系统框架对该问题进行详细的案例研究和性能评估.

3 案例研究:线程间不必要锁竞争问题

3.1 问题描述

在多线程程序中,锁同步机制常用来保证临界区中共享资源的互斥访问.例如,某一个线程正在访问它的临界区资源,那么其他所有的线程将不得不推迟对临界区资源的访问,直到该线程完成对临界区的访问.这种模式串行化了临界区的访问,因此保证了同一共享资源访问的互斥性与正确性.

但是,由于各种各样的原因,程序动态执行过程中,多个线程同时访问临界区资源时并不一定会造成冲突访问,因此,这种锁竞争所带来的串行化执行其实是不必要的,即不必要锁竞争(unnecessary lock contention, ULC),并造成程序性能上的下降.图 5 给出了一个简化的典型例子,当线程局部变量 *local_variable* 的值为假、多个线程同时访问该临界区时,各线程临界区其实并没有共享访存行为,因此在动态执行时,临界区锁的获取和释放操作在局部变量 *local_variable* 为假的情况下是没必要的.该问题可进一步拓展为:线程间的两个临界区有着非冲突访存行为,例如,图 5 中的无共享访存行为、只有共享读操作、不同的共享读写访存行为.

```
LOCK(L);
if (local_variable)
   shared_variable++;
UNLOCK(L);
```

Fig. 5 A generic model for the unnecessary lock contention.

图 5 一个通用的不必要锁竞争模型

3.2 程序记录文件转化

为了在 PerfPlay 的框架下解决线程间的不必要锁竞争问题,我们对步骤 2 进行了特定地分析及问题消除.为了方便描述,我们作如下定义:

C 表示某一临界区;$C.S_{rd}$ 表示临界区 C 中所有共享读操作集合;$C.S_{wr}$ 表示临界区 C 中所有共享写操作集合.

具体来说,我们的策略进一步描述如下:

1) 分析记录文件.按照算法 1 中的规则求两个相邻的含有直接依赖关系的临界区的读写集的 3 种情况(如算法 1 中行①所示)是否相交.如果交集为空,说明找到了一组不必要锁竞争对,否则,则不是一组不必要的锁竞争对.用算法 1 遍历整个记录文件,识别出程序执行文件中所有的不必要锁竞争对.

算法 1. ULC 确认算法.

输入:$\langle C_1, C_2\rangle$,2 个临界区;

输出:Retval.

FALSE, $\langle C_1, C_2\rangle$ consists of an ULC,

TRUE, $\langle C_1, C_2\rangle$ is not an ULC.

① if $C_1.S_{rd} \sqcap C_2.S_{wr} = \varnothing$ and $C_2.S_{wr} \sqcap C_1.S_{rd} = \varnothing$ and $C_1.S_{wr} \sqcap C_2.S_{wr} = \varnothing$

② return TRUE

③ else

④ return FALSE.

2) 消除记录文件中的不必要锁竞争.分析程序时序文件,消除记录文件中识别出的不必要锁竞争对维持的依赖偏序时序关系.如图 6 所示:

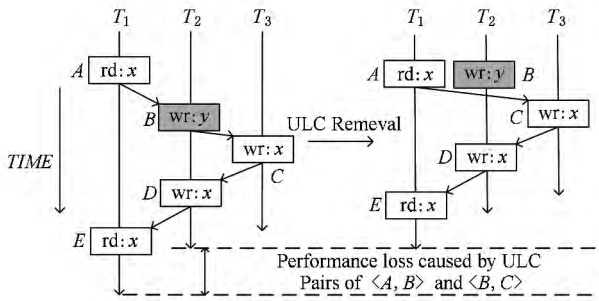


Fig. 6 The elimination of false inter-thread dependence caused by ULC based on the original program trace.

图 6 消除原始记录文件中所有的 ULC 依赖关系

原始程序执行轨迹的临界区时序偏序关系为

$$A \prec B; B \prec C; C \prec D; D \prec E; \tag{2}$$

式(2)中,存在两个不必要锁竞争对$\langle A, B\rangle$和$\langle B, C\rangle$,因此它们之间的偏序时序关系应该删除,即修改的依赖时序关系为

$$C \prec D; D \prec E. \tag{3}$$

但是上述时序违反了 RLSC 约束条件.尽管在当前执行下,$A \parallel B$ 及 $B \parallel C$;但是临界区 A 和 C 访问了冲突的同一共享内存空间.按照 RLSC 策略,它们之间的时序关系必须得到保证.为了避免这种情况下程序执行的稳定性,在消除过程中,PerfPlay 会

自动检测所消除不必要锁竞争对的偏序关系的传递性(即若 $A \prec B$ 和 $B \prec C$,那么 $A \prec C$). 若消除的不必要锁竞争对存在偏序关系的传递性,传递的锁竞争对也会按照算法 1 重新进行识别,进而判断是否该维持它们之间的时序关系. 例如,A 和 C 不构成一个不必要锁竞争对,因此,它们的时序关系必须要维持,因此,最终修改后的依赖时序偏序关系为

$$A \prec C;\ C \prec D;\ D \prec E. \tag{4}$$

3.3　性能定量分析

基于 PerfPlay 框架,我们亦可以对 ULC 按照代码产生源进行归类,进而定量分析其造成的性能问题. 即程序源码层中有哪些代码区域造成了 ULC,它们都各自造成多大的性能损耗.

1) ULC 性能度量标准. 图 7 描述了动态执行过程中某个 ULC$\langle A,B\rangle$两种可能的性能度量情况:

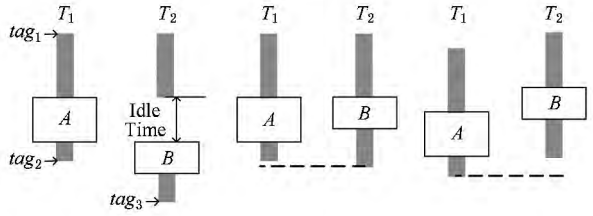


Fig. 7　Two possible schedules after the optimization for the performance measurement of each ULC.

图 7　ULC 优化后程序可能的调度情况

标识临界区 A 前驱程序段的第 1 个指令为 tag_1,后驱程序程序段的最后一个指令为 tag_2,临界区后驱程序段的最后一个指令为 tag_3. 为了能衡量图中两种调度情况,对单个 ULC 优化后性能度量标准为

$$\Delta T_{\mathrm{ULC}} = \Delta \mathrm{MAX}(T_{tag_2}, T_{tag_3}) - \Delta T_{tag_1}, \tag{5}$$

其中,T_{tag} 表示程序执行到标识 tag 时系统当前的时间戳,而 MAX 函数的功能为求一组数据集的最大值.

2) ULC 性能定量分析. 获得动态执行序列中单个 ULC 的性能优化值后,进而我们可以按照代码产生源对这些分散的 ULC 进行合并,获得唯一的 ULC 集$\{\mathrm{ULC}_1, \mathrm{ULC}_2, \cdots, \mathrm{ULC}_n\}$. 在该 ULC 集合中,任意的两个元素都不可能是由同一代码段生成. 基于此,我们可以定量分析每个代码段是否值得优化.

$$P_i = \frac{\Delta T_{ULC_i}}{\sum_{j=1}^{n} \Delta T_{ULC_j}}, \tag{6}$$

其中,P 值表示该 ULC 占所有 ULC 造成性能损耗的比例,P 值越大则表示该 ULC 越值得优化.

4　实验结果

PerfPlay 采用 C/C++语言实现,现阶段只能识别由 pthread 同步原语实现的多线程程序. 该部分所有程序测试的物理机配置为:双处理器 4 核心 Intel Xeon CPU E5310;CPU 工作频率 1.6 GHz;4 GB RAM. 运行的操作系统为 CentOS5.6,X86_64.

4.1　测试程序

我们使用了不同的多线程程序来测试 PerfPlay 重放系统的不同特征. 这些应用包括服务器程序(openLDAP[23]和 mysql[24])、桌面应用(pbzip2[25], transmissionBT[26]和 handbrake[27])及 PARSEC[28] 基准程序. 表 1 给出了关于它们的描述与测试参数:

Table 1　A Summary of the Benchmarks and the Corresponding Options

表 1　测试程序集及相应的测试参数

Applications	Description	Test Options
openLDAP	A lightweight directory access protocol server	2 000 entries searching
mysql	An open source database	1 000 queries, 5 iterations
pbzip2	A parallel data compression software	256 M file compression
transmissionBT	A BitTorrent client	300 M file download
handbrake	A video format convertor	256 M DVD format into MP4 format H.264 codec, 30FPS
PARSEC Benchmark	A benchmark suite with multithreaded programs	simlarge

4.2　记录开销

基于 PerfPlay 框架我们分别用全局记录和选择性记录的方式记录了 openLDAP,mysql,pbzip2, transmissionBT 及 handbrake 程序的执行轨迹,每个程序均启动了两个线程. 表 2 列出了各记录文件按照全域记录和选择性记录的方式进行记录的生成速率. 例如,openLDAP 记录文件在全域记录和选择性记录方式下的生成速率分别为 101 MBps 和 6.4 MBps,

选择性记录方式使得 openLDAP 记录文件生成速率下降 93.7%. 其他应用 mysql, pbzip2, transmissionBT, hanbrake 生成速率也分别不同程度地下降 91.6%, 97.5%, 97.7%, 92.6%. 相比于全域记录方式, 选择性记录方式使得记录文件生成速率平均下降 93.85%.

Table 2 The comparison of Recording Overhead with Complete Recording and Selective Recording

表 2 全域记录与选择性记录方式的记录开销对比

Applications	Recording Mode	
	Complete/MBps	Selective/MBps
openLDAP	101	6.40
mysql	169	14.1
pbzip2	51	1.25
transmissionBT	92	2.11
handbrake	121	8.93
Average	106.8	6.56

由此可见, 选择性记录方式极大地减少了全域记录高额的记录开销, 并使得 PerfPlay 系统的记录过程变得更高效、更简单.

4.3 性能保真度

为了评测 PerfPlay 系统的性能保真度, 我们对比了不同的重放技术, 包括 CoreDet, Kendo, PerfPlay w/o RLSC(不使用 RLSC 策略时的原始并行重放). 图 8 统计了 PARSEC 各多线程应用重放时的运行时间, 每个程序被重放 10 次.

我们发现, 几乎对所有程序, 各重放技术的重放开销依次为: CoreDet > Kendo > PerfPlay w/RSC ≈ PerfPlay w/o RSC. 1) 由于重放工具 CoreDet, Kendo 和 PerfPlay w/ RLSC 提供了确定性执行技术, 因而它们均能提供稳定的性能重放. 例如, 在没有采用 RLSC 时, 由于程序运行时复杂的线程交错关系, PerfPlay w/o RLSC 表现出很大的重放性能波动(即不稳定性), 这说明 RLSC 策略在很大程度上保证了重放执行性能的稳定. 2) 相较于原始执行的并行重放(PerfPlay w/o RLSC), CoreDet 和 Kendo 均有不同程度的额外开销, 带来了不精确的原始重放执行, 主要是因为它们所采用的确定性技术因不同的功能或性能需求强制延长了程序的执行. 而 RLSC 基于程序调度的确定性策略严格地遵守了原始执行, 因此, 其性能几乎与原始执行的并行重放一致, 这又说明了 RLSC 策略严格地保证了程序性能的精确性. 结合图 8 及上述分析可知, RLSC 策略保证了并行重放系统程序重放执行时的性能保真度.

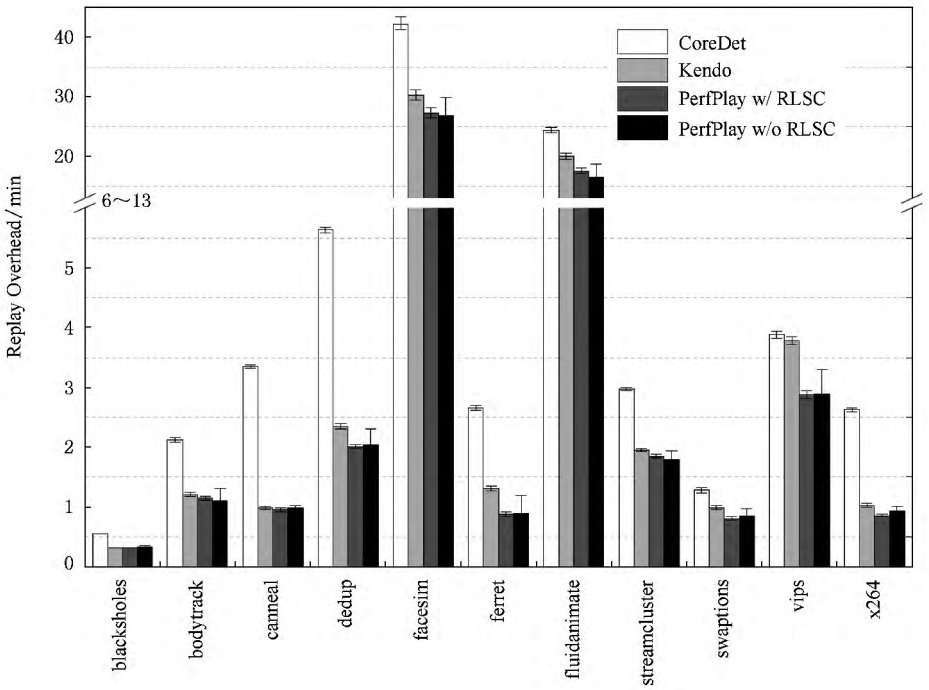


Fig. 8 The stability and precision of performance for different replay techniques.

图 8 不同重放技术的性能稳定性与精确性

4.4 有效性验证

为了验证 PerfPlay 性能重放系统的有效性, 我们对"不必要锁竞争"问题进行了案例研究. 图 9 描述了消除记录文件中"不必要锁竞争"后各程序可优

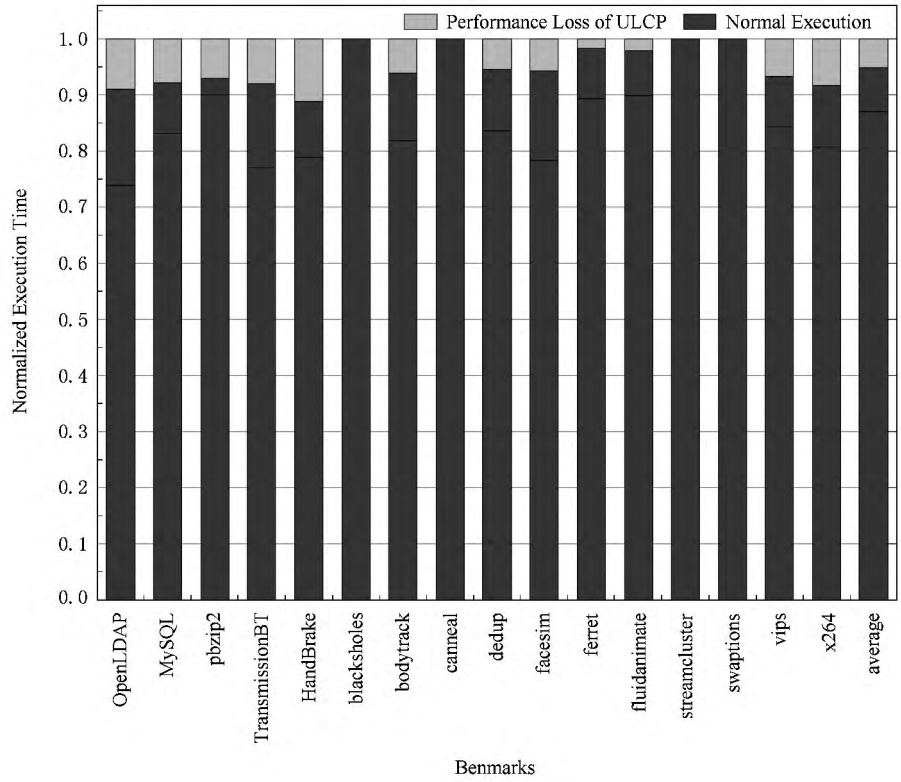


Fig. 9　The normalized execution time through replaying the traces with and without ULCs.

图 9　ULC 优化前后的程序性能比例分布

化的性能空间. 从图 9 可以看出,各程序中表现出不同的可优化空间. 例如,blacksholes, canneal, streamcluster **和** swaptions **几乎没有任何可优化的空间,因为它们要么是没有使用锁操作,抑或是使用了少量的锁且正确使用了这些锁. 但对于其他应用来说,** ULC **造成了不同程度的(**1.6%～11%**)性能损耗,平均来说,**ULC **造成了** 5.1% **的性能下降.**

结合 4.3 **节的性能定量分析,我们利用** PerfPlay **性能重放系统分析了** pbzip2 **中的** ULC **性能问题,并对其进行了验证,查证** PerfPlay **性能重放系统是否找到了一个真实的** ULC **代码源. 图** 10 **列出了一个定量分析出的例子. 在该例中,若压缩文件最后一个块被线程读取之后,此时** $fifo->file=1$ **及** $producerDone=1$**,并且程序开始进入退出阶段. 此时,在多个线程都需要退出时,上述代码将会在每个线程中执行,并生成如下简化后的代码:**

```
lock(mu);
read(fifo->empty);
lock(mu1);
read(producerDone);
unlock(mu1);
unlock(mu);
```

该段代码串行化了程序退出,由于嵌套锁的关系,整个退出过程除了造成复杂的串行开销外同时还带来了额外的锁操作(即获取和释放)开销. 我们用 pbzip2 **启动了** 8 **个线程来压缩一个** 64 MB **相对较小的文件,并利用内部定时器对该** ULC **问题进行重新测试和量化分析,发现该** ULC **造成约** 10.54% **的程序性能下降(包括串行开销及锁操作开销等).**

```
void *consumer(void *q){
⋮
2109: pthread_mutex_lock(&mu);
2122: if (fifo->empty && syncGetProducerDone()==1)
2124: pthread_mutex_unlock(&mu);
⋮
}
int syncGetProducerDone(){
533: int ret;
534: pthread_mutex_lock(&muDone);
535: ret=producerDone;
536: pthread_mutex_unlock(&muDone);
537: return ret;
538: }
```

Fig. 10　A real ULC problem from pbzip2.

图 10　一个 pbzip2 **中的** ULC **性能问题例子**

5 总 结

近年来,多线程程序中的性能 bug 问题越来越突出,越来越重要.传统用于调试并发错误的记录/重放系统由于重放开销与性能不精确性等问题已不再适于多线程程序性能 bug 的调试.基于此,本文提出了一种可用于多线程程序性能分析的改进型记录/重放系统—PerfPlay,并从性能信息记录、性能重放执行、基于重放执行的性能调试等多个层面对传统记录/重放系统进行了深入的分析与改进.通过对经典的"线程间不必要锁竞争"问题的案例研究,进一步验证了性能重放系统 PerfPlay 在性能 bug 发掘上的有效性. PerfPlay 系统首次实现了重放技术在性能 bug 调试上的应用,未来如何利用 PerfPlay 重放系统去探索并解决多线程程序中更多的性能问题是一个值得深入研究和探讨的课题.

参 考 文 献

[1] Nielson F, Nielson H R, Hankin C. Principle of Program Analysis [M]. Berlin: Springer, 1999: 1-27

[2] Narayanasamy S, Wang Z, Tigani J, et al. Automatically classifying benign and harmful data races using replay analysis [C] //Proc of the 28th Conf on Programming Language Design and Implementation. New York: ACM, 2007: 22-31

[3] Samak M, Ramanathan M K. Trace driven dynamic deadlock detection and reproduction [C] //Proc of the 19th Symp on Principles and Practice of Parallel Programming. New York: ACM, 2014: 29-42

[4] Park S, Lu S, Zhou Y. CTrigger: Exposing atomicity violation bugs from their hiding places [C] //Proc of the 14th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2009: 25-36

[5] Patil H, Pereira C, Stallcup M, et al. PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs [C] //Proc of the 8th Int Symp on Code Generation and Optimization. New York: ACM, 2010: 2-11

[6] Altekar G, Stoica I. ODR: Output-deterministic replay for multicore debugging [C] //Proc of the 22nd Symp on Operating Systems Principles. New York: ACM, 2009: 193-206

[7] Liu Lei, Huang He, Tang Zhimin. High efficient memory race recording scheme for parallel program deterministic replay under multi-core architecture [J]. Journal of Computer Research and Development, 2012: 49(1): 64-75 (in Chinese)
(刘磊, 黄河, 唐志敏. 支持多核并行程序确定性重放的高效访存冲突记录方法[J]. 计算机研究与进展, 2012, 49(1): 64-75)

[8] C++ STANDARDS COMMITTEE. C++ final draft international standard [S]. New York: ISO/IEC, 2011

[9] C STANDARDS COMMITTEE. Committee draft: Programming languages [S]. New York: ISO/IEC, 2010

[10] Jin G, Song L, Shi X, et al. Understanding and detecting real-world performance bugs [C] //Proc of the 33rd Conf on Programming Language Design and Implementation. New York: ACM, 2012: 77-88

[11] Bergan T, Anderson O, Devietti J, et al. CoreDet: A compiler and runtime system for deterministic multithreaded execution [C] //Proc of the 15th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2010: 53-64

[12] Olszewski M, Ansel J, Amarasinghe S. Kendo: Efficient deterministic multithreading in software [C] //Proc of the 14th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2009: 97-108

[13] Hall R J. Call path profiling [C] //Proc of the 14th Int Conf on Software Engineering. New York: ACM, 1992: 296-306

[14] Han S, Dang Y, Ge S, et al. Performance debugging in the large via mining millions of stack traces [C] //Proc of the 34th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2012: 145-155

[15] Rajwar R, Goodman J R. Speculative lock elision: Enabling highly concurrent multithreaded execution [C] //Proc of the 34th Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2001: 294-305

[16] Rajwar R, Goodman J R. Transactional lock-free execution of lock-based programs [C] //Proc of the 10th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2002: 5-17

[17] Narayanasamy S, Pokam G, Calder B. Bugnet: Continuously recording program execution for deterministic replay debugging [C] //Proc of the 32nd Int Symp on Computer Architecture. New York: ACM, 2005: 284-295

[18] Xu M, Bodik R, Hill M D. A "flight data recorder" for enabling full-system multiprocessor deterministic replay [C] //Proc of the 30th Int Symp on Computer Architecture. New York: ACM, 2003: 122-135

[19] Zhou J, Xiao X, Zhang C. Stride: Search-based deterministic replay in polynomial time via bounded linkage [C] // Proc of the 34th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2012: 892-902

[20] Lee D, Said M, Narayanasamy S, et al. Offline symbolic analysis to infer Total Store Order [C] //Proc of the 17th Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2011: 357-358

[21] Bell Labs. SPIN Verification Tool [CP/OL]. 1991 (2013-05-04) [2014-05-12]. http://spinroot.com/spin/whatispin.html

[22] Kemper B. PIN Binary Instrumentation Framework [CP/OL]. 2005 (2014-03) [2014-05-12]. https://sites.google.com/site/pintutorial/home/asplos2014

[23] OpenLDAP Foundation. An open source implementation of the Lightweight Directory Access Protocol [CP/OL]. 1998 (2011-11-23) [2014-05-12]. http://www.openldap.org/

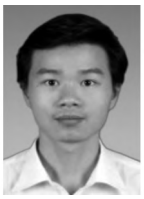
[24] Oracle Corporation. MySQL relational database management system [CP/OL]. 1995 (2014-09-23) [2014-05-12]. http://www.mysql.com/

[25] Gilchrist J. A parallel bzip2 compressor [CP/OL]. 2008 (2012-06-06) [2014-05-12]. http://compression.ca/pbzip2/

[26] Lee J. Transmission BitTorrent Client [CP/OL]. 2005 (2013-08-08) [2014-05-12]. http://www.transmissionbt.com/

[27] HandBrake Community. Handbrake Video Transcoder [CP/OL]. 2003 (2013-05-18) [2014-05-12]. http://handbrake.fr/

[28] Bienia C. PARSEC Benchmark Suite [CP/OL]. 2008 (2009-02) [2014-05-12]. http://parsec.cs.princeton.edu/



Zheng Long, born in 1987. PhD candidate. His current research interest mainly focuses on program analysis.

Liao Xiaofei, born in 1978. Professor and PhD supervisor. Senior member of China Computer Federation. His current research interests include runtime optimization, system software and distributed systems.

Wu Song, born in 1975. Professor and PhD supervisor. Senior member of China Computer Federation. His main research interests include cloud computing, data center management and virtualization.

Jin Hai, born in 1966. Professor and PhD supervisor. Fellow member of China Computer Federation. His main research interests include big data analysis, cloud computing, cloud security and data center management.