北京大学学报(自然科学版) 第 56 卷 第 6 期 2020 年 11 月
Acta Scientiarum Naturalium Universitatis Pekinensis, Vol. 56, No. 6 (Nov. 2020)
doi: 10.13209/j.0479-8023.2020.100

高性能处理器中干扰公平队列I/O调度器

隋岩 叶嘉成 杨春† 佟冬

北京大学信息科学技术学院 北京 100871; †通信作者, E-mail: yangchun@mprc.pku.edu.cn

摘要 高性能处理器和系统需要高存储带宽和高效的外部I/O处理, 需要同时服务吞吐率密集应用和延迟敏感应用, 给多道程序计算机系统和多租户模式的超级计算机系统的公平性带来巨大的挑战。针对这两类应用共享SSD(solid-state disks)等可并发的存储设备问题, 开发一款基于队列的干扰公平(interference fair queueing, IFQ)调度器。在Linux操作系统实现IFQ调度器, 并与其他调度器进行对比, 包括Linux的CFQ调度器、STF调度器、MFAP的时间片流转调度器和MFAP的短时间片流转调度器。基于合成工作集、访问踪迹工作集和真实应用工作集的结果显示, IFQ调度器可以同时保证公平性和响应延迟。

关键词 存储调度器; 干扰公平; 固态硬盘

An Interference Fair Queueing I/O Scheduler under High Performance Processors

SUI Yan, YE Jiacheng, YANG Chun†, TONG Dong

School of Electronics Engineering and Computer Science, Peking University, Beijing 100871;
† Corresponding author, E-mail: yangchun@mprc.pku.edu.cn

Abstract High performance processors and systems require high storage bandwidth and efficient external I/O processing. Many systems using high performance processors server a mixture of fully backlogged users, which continuously demand resources, and non-fully backlogged users. This presents challenges for fair resource management in multi-programmed computer systems and multi-tenant super computer systems. This paper develops a new storage I/O scheduler IFQ (interference fair queueing) for parallel accessible devices, such as SSDs. IFQ is implemented in Linux and compared with several existing I/O schedulers—Linux CFQ, STF, MFAP scheduler and MFAP scheduler with short time-slices. Results on synthetic I/O benchmarks, trace benchmarks, and real-world benchmarks demonstrate that only IFQ can achieve both fairness and high responsiveness.

Key words storage scheduler; interence fairness; solid-state disks

近年来高性能处理器得到巨大发展, 单核处理器性能和并行核数随之提升。与处理器性能的巨大提升相比, 存储I/O的性能提升不明显, 这进一步加剧存储墙问题(memory/storage wall problem)。高性能处理器及系统需要高存储带宽和高效的外部I/O处理, 随着超级计算机(super computer)的出现, 存储带宽和外部I/O资源的分配, 特别是公平分配问题, 也日益显著[1−3]。

资源分配是共享计算机系统的核心内容之一[4]。在相同的用户之间分配资源是一个平凡问题, 为了在不同用户之间公平地分配资源, 学者们开展了很多公平性研究。加权最大最小(weighted maximize min, WMM)按照各个用户的不同权重, 等比例地分配资源。比例分配(proportional sharing, PS)和公平加权队列(weighted fairness queue, WFQ)等WMM的实现广泛用于处理器[5]、内存[6]、网络[7−8]和存储[9]等资源的分配。为了在不同需求的用户之间同时分配多种资源, Ghodsi等[10]提出支配资源公平

国家科技重大专项(2018ZX01029101)资助
收稿日期: 2019−11−22; 修回日期: 2020−02−06

(dominant resource fairness, DRF)分配策略, 对系统中的资源按照稀缺程度进行公平分配。基于 DRF, 人们进一步提出一些资源分配方法和公平策略[11–12]。Dolev 等[13]和 Wang 等[14]将系统的性能瓶颈资源加入 DRF 方法中予以考虑。Gutman 等[15]给出一个 DRF 分配的多项式时间分配算法。

以上公平性研究都是针对吞吐率密集的满负荷(fully backlogged)用户, 但实际上很多延迟敏感用户并非永远满负荷地需求资源。这些用户在某些时刻会出现不需求资源的空闲阶段(idle time)。这种空闲主要来自两方面: 一方面是使用者交互过程中的思考时间(thinking time), 例如网页浏览器和游戏; 另一方面是对其他资源的阻塞式访问, 例如在用户启动和文件压缩等过程中, 用户应用顺序阻塞式地依次需求计算资源和存储资源。针对满负荷用户的公平性策略对非满负荷的用户是不公平的。非满负荷用户并不始终需求资源, 然而当他们有资源需求时, 依然需要和其他用户“公平”地共享资源。

为了解决这些问题, 研究者引入一个新的公平指标——干扰公平[16]。其基本想法是, 用户 A 对用户 B 的干扰, 应等同于用户 B 对用户 A 的干扰。为了在存储系统中有效且高效地保证干扰公平, 研究人员进一步提出 MFAP[16], 根据各个用户访问存储资源的时间来评估消耗的资源量, 动态地分配各个用户竞争资源时的权重, 并通过不同尺寸时间片的轮转方法来实现。但是, 此方法的缺陷是, 一旦一个用户耗尽时间片, 必须等待下一个时间片的到来, 会导致非常高的 I/O 请求响应延迟, 严重地影响用户体验。

响应延迟保证通常基于公平队列方法实现, 广泛应用于网络[17]、存储 I/O 调度[18]和 GPU 调度[19–20]等。此方法支持高细粒度的请求交错调度, 同时使用虚拟时间戳来保证公平性。当一个用户发出请求并得到服务时, 其虚拟时间戳会相应后移, 每次调度器都选择虚拟时间戳最靠前的用户进行服务。这种细粒度调度算法的主要问题是破坏了请求流的空间局部性, 因此会给硬盘驱动器(hard disk drive, HDD)带来额外的性能开销。幸运的是, 在固态硬盘(solid-state disk, SSD)上, 破坏请求流的空间局部性对系统性能影响不大。已有的 MFAP 资源消耗评估方法均针对 HDD, 而 SSD 设备有不同于 HDD 设备的独特性, 包括并行访问和性能不稳定等, 使得 MFAP 基于访问存储设备时间的评估资源消耗方法并不适用于 SSD 设备。

针对以上问题, 本文开发一个基于队列的干扰公平(interference fair queueing, IFQ)调度器, 可以在 SSD 设备下同时保证请求的响应延迟和干扰公平。与现有的 MFAP 干扰公平调度器相比, IFQ 调度器主要有两点改进: 使用队列调度算法, 可以有效地保证请求响应延迟; 使用访问数据量而不是访问时间来评估资源消耗, 可以支持 SSD 的并发访问和性能不稳定等特征。

1 研究动机

1.1 SSD 设备的特性

我们从 3 个角度分析 SSD 设备的特性: 并发访问特性、随机访问特性和性能不稳定特性。

一个典型的 SSD 设备存在若干通道(channel), 各通道内可能存在多个通路(plane)供数据包传输, 因此 SSD 的访问存在内置的并行性。我们通过一个实验来展示这种并行性。为了尽可能地降低干扰, 本文使用 direct I/O 技术跳过操作系统的页缓存, 同时使用 Linux 的 noop 调度器来减少调度器带来的干扰。分别测试典型 SSD 设备下, 4 和 128 KB 数据的读操作性能, 结果见图 1。可以看出, 随着并发读进程数量的提升, 系统吞吐率随之提升。

SSD 设备还具有随机访问的特性, 与 HDD 设备相比, SSD 设备在服务随机访问请求流时, 只损失较少的性能。由于 SSD 设备控制器的复杂性, 其吞吐率并不恒定。例如, SSD 设备的垃圾回收机制(garbage collection)使得其写入效率有非常大的变化, 即一段时间内有很高的写入速度, 之后速度会大幅度降低。

1.2 时间片轮转调度和队列调度

很多 I/O 调度器都基于时间片流转, 如 Linux

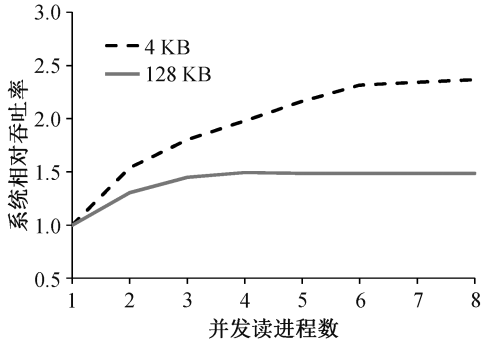


图 1 SSD 设备的并行访问特性示意图

Fig. 1 Parallel access characteristics of SSD devices

的 CFQ 调度器[21]和 MFAP 调度器[16]等。这些调度器给予不同的应用以不同大小的时间片，之后依次调度这些时间片来服务应用请求。如果一个应用耗尽其时间片就需要等待其他应用，直到再次轮转到它为止。在这段时间内，应用的请求无法得到响应，影响用户体验，如图 2(a)所示。

为了解决这一问题，一些调度器选择较短的时间片长度，如图 2(b)所示。这种方法只能缓解而不能解决该问题。更严重的是，较短的时间片会导致更多的边界问题，带来公平性的降低。也有一些调度器使用请求交错的轮转调度(图 2 (c))，但这种方法的问题是，当一个应用的时间片耗尽后，其请求延迟依然无法得到保证。

基于公平队列的调度能够在细粒度上保证公平性，使得来自不同应用的请求被间隔地调度，如图 2(d)所示。这种细粒度的公平性有 3 个优势：不需要等待时间片的轮转，各个应用请求的响应延迟都可以得到保证；支持多个应用同时发出请求，从而充分地利用支持并发访问的设备(如 SSD 等)；可以有效地解决 SSD 设备访问性能变化的问题。

1.3 长尾延迟问题

交互式的桌面应用和交互式的网络服务需要稳定的低响应延迟来吸引和维系用户群，因此这些交互式场景对尾部延迟非常敏感[22–23]。通常认为，长尾延时越低，服务质量越高。

我们使用 UMass 的 4 个延迟敏感服务的 I/O 访问踪迹进行性能测试，分别在 HDD 和 SSD 设备下，评估 MFAP 算法 99.9%的最差尾部延迟和平均延迟的比例(图 3)。可以看出，SSD 设备下的长尾问题比 HDD 设备更严重，这是由于不同设备下的长尾延迟主要取决于时间片轮转算法，因此比较接近，而 SSD 的平均延迟远小于 HDD 设备。

2 公平性调度算法

2.1 公平性原则

为了比较分配方式的公平性，受 DRF 和 RRF 的启发，MFAP 给出以下 4 条原则，并认为对于吞吐率密集和延迟敏感的应用之间的资源分配应该满足这些原则[16]。

1) 共享激励原则：各个用户共享资源的性能不能比单独使用均等分配的资源更差。特别地，对于 n 个用户，各个用户享有的资源性能不能比独享 1/n 资源差[10,16,24]。

2) 策略最优原则：各个用户无法通过虚构自身特性来获得性能提升[10,16,24]。

3) 性能单调原则：当系统资源或系统负载变化时，各个用户享有性能的变化趋势一致[10,16,24]。

4) 多劳多得原则：多劳多得是非常重要的公平性原则[16,24]。

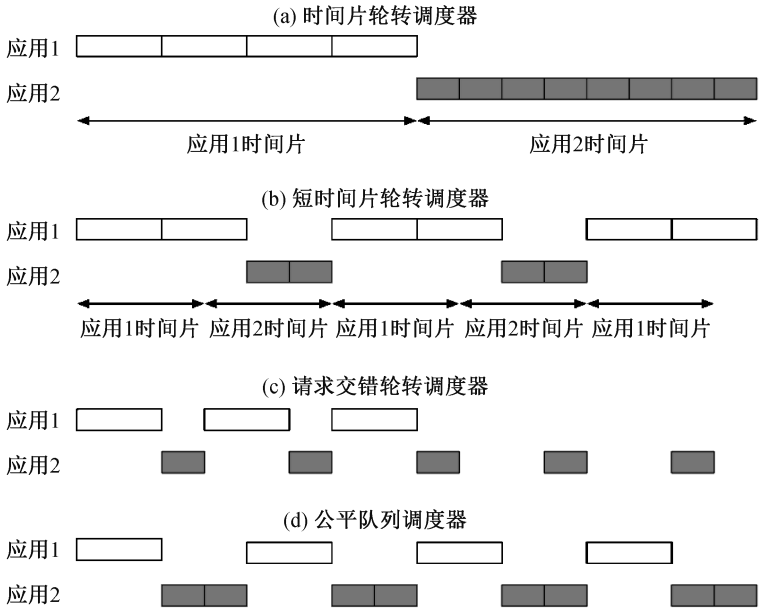


图 2　不同类型的调度器下公平性和响应延迟示意图

Fig. 2　Fairness and response delay under different types of schedulers

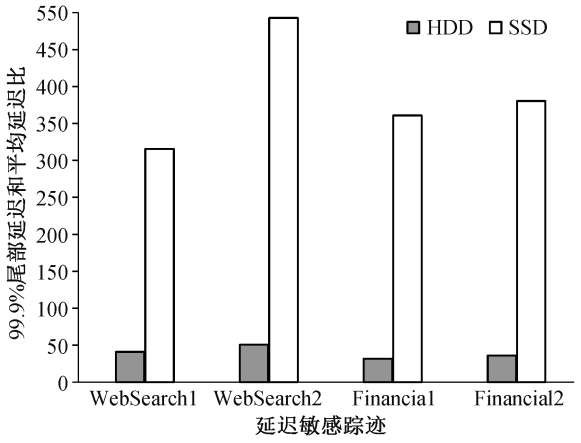


图 3 MFAP 算法在不同设备下 99.9%尾部延迟和平均延迟对比

Fig. 3 99.9% tail delay and average delay of MFAP in different devices

从总体上看, 4 个原则最终都可能归因于经济学或博弈论的公平问题, 它们符合常识的直觉。如 DRF 中讨论的, 策略最优和共享激励原则在商业化数据中心的环境下对保证不同付费用户之间的公平性非常重要。策略最优避免了严重的虚构特征的问题, 例如用户在他们的代码中加入无限循环, 人为地膨胀资源消耗, 以便提升其性能[5]。此外, 满足共享激励原则的任何策略还提供性能隔离, 因为它有效地保证了每个用户的最小分配性能不受其他用户需求的影响。性能单调是一个非常直观的公平性概念。多劳多得原则奖励客户消耗较少的资源, 如果用户消耗更少的资源, 将对系统贡献更多, 其他用户就可以共享更多资源, 并在性能方面受益。

2.2 MFAP 调度算法

MFAP 调度算法的核心问题是如何获得用户的统计信息。从直观上理解, MFAP 需要在各个用户的工作集执行之前获得其详细信息, 以指导 MFAP 的调度, 这在真实环境中不现实, 也不可能。为了解决这一问题, MFAP 使用一种推测式的调度算法, 将整个系统的执行过程划分为若干等长的时间片段, 使用前一时间片段的信息指导下一时间片段的调度情况。

MFAP 的干扰公平存储访问调度器面向 HDD 设备, 因此在没有 RAID 的情况下, MFAP 模型中只有一个资源实例, 所以 MFAP 调度算法基于不同用户使用存储资源的时间来评估用户的资源消耗。

基于 MFAP 策略的不同权重, 我们分配不同的时间片给各个用户。例如, 如果检测到两个用户分别使用存储资源 10 和 20 秒, MFAP 调度算法就给两个用户分别分配 200 和 100 毫秒的时间片。为了准确地估计用户使用存储资源的时间, MFAP 会统计各个用户使用时间片的数量, 并且标记未使用完的时间片, 以备计算。

3 基于队列的干扰公平调度器

3.1 干扰公平分配策略

干扰公平需要保证各个用户之间相互的干扰程度一致。简单地定量计算这些干扰, 会给 n 个用户共享资源的系统带来 $O(n^2)$的计算开销, 难以有效地实现。因此, 我们提出一种干扰公平分配策略, 计算每个用户实际需求资源时的资源分配比率。当两个用户竞争资源时, 各自的分配比率与其消耗的资源成反比。例如, 与用户 A 相比, 用户 B 消耗两倍数量的资源, 则干扰公平分配策略在两个用户竞争资源时, 会按照 2:1 的比例在用户 A 与 B 之间分配资源。

以一个包含 6 个资源实例(如 6 个处理器等)和 3 个用户的系统为例, 整个执行过程持续 12 个单位的时间, t=0 执行过程开始, t=12 执行过程结束。如果 1 个用户在 1 个时间单位内使用 1 个资源实例, 干扰公平分配策略则定义此用户消耗了 1 个单位的资源, 即资源消耗=资源实例数×使用时间。因此, 有 6×12 = 72 个单位的资源消耗可供 3 个用户分配。假设用户 A 和 B 是延迟敏感用户, 用户 C 是吞吐率密集的, 则用户 A 在 t=0 时开始需求资源, 并且在消耗 18 个单位的资源后结束; 用户 B 在 t=2 时开始需求资源, 也在消耗 18 个单位的资源后结束; 用户 C 从 t=0 时开始一直需求资源。

计算得出用户 A, B 和 C 分别消耗 18, 18 和 36 个单位的资源, 在干扰公平分配策略下, 3 个用户分配资源的权重被分配为 2:2:1。这一分配资源的权重只有在用户互相竞争使用资源时才有效, 因此并不意味着资源的分配比例永远是 2:2:1, 或用户 A 获得 40%的资源保留。

图 4 为一个具体的分配示例。在时间段 t=[0, 2], 用户 A 与 C 竞争资源, 其资源分配比例为 2:1。A 分配 4 个资源实例, C 分配两个。在这段时间内, A 消耗 8 个单位的资源, C 消耗 4 个单位。在 t=2, B 开始需求资源, 在时间段 t=[2, 6.17], 用户 A, B 与 C 竞争资源, 资源分配比率为 2:2:1, 因此 A 和 B 各自被分配 2.4 个资源实例, 而 C 被分配 1.2 个。在这

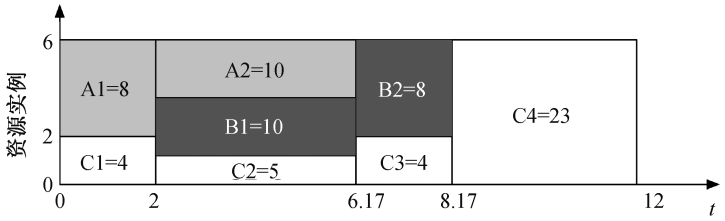


图 4　干扰公平资源分配示例

Fig. 4　Example of interference fair resource allocation

段时间，A 和 B 分别消耗 10 个单位的资源，C 消耗 5 个。之后，A 完成工作退出资源竞争，在时间段 t=[6.17, 8.17]，用户 B 和 C 竞争资源，分配比例为 2:1，B 被分配 4 个资源实例，C 被分配两个。这段时间 B 消耗 8 个资源并完成工作，C 消耗 4 个。在 t=[8.17, 12]，用户 C 独占资源，消耗 23 个单位的资源。

3.2 资源消耗评估方法

干扰公平调度实现的核心是资源消耗的评估，在 MFAP 算法中，对于 HDD 设备，可以用应用访问 HDD 设备的时间来评估开销。但是，在 SSD 设备上使用此方法却存在以下矛盾：一方面 SSD 设备支持并发访问，另一方面，SSD 设备的吞吐率并不固定。为了解决这一问题，我们使用数据访问量来评估应用对 SSD 资源的消耗，即认为应用对 SSD 资源的消耗与其请求数据量成正比。这一方法也应用在其他针对 SSD 设备公平性的研究中[18]。

与 MFAP 算法类似，IFQ 也使用推测式的调度策略，即将整个系统的执行过程划分为若干等长的时间片段，使用前一时间片段的资源消耗信息，指导下一时间片段的调度情况。

3.3 加权公平队列算法选择

加权公平队列通常使用虚拟时间(virtual time)的方法实现。维护虚拟时间的算法主要有以下 3 种：Min-SFQ(D)，Max-SFQ(D)和 4-Tag SFQ(D)。Max-SFQ(D)和 4-Tag SFQ (D)算法更多地将来自不同用户的请求全部发给存储设备，而不是在调度器中进行相应的缓存处理[18]。

在基于加权公平队列的实现中，我们尽可能地将请求缓存在调度器层面，以便减少 SSD 设备自身特性带来的影响，因此选择基于 Min-SFQ(D)的加权公平队列实现方式。如果检测到两个应用分别发出 100 和 200 MB 的请求，就给两个应用分别分配 2 和 1 的权重[16]。

在 SFQ(D)算法中，更大的深度值可以保证更好的请求并行性，但也会使更多的请求从调度器发送给存储设备。本文将 SFQ(D)算法的深度(depth, D)配置为 8，我们认为 8 的配置可以充分地发掘典型 SSD 设备的并行性。

4 实验评测

本文实验平台基于 3.18.34 内核的 Centos 6.5 系统。此系统安装在一台浪潮 NF5280 服务器上，服务器拥有 4 个 Intel 四核 E5620 处理器和 64 GB 的内存，使用一块 SATA HDD 硬盘存放操作系统，一块 120 GB 三星 750 EVO SATA3 SSD 用来存放测试文件，用以保证实验不受系统中其他进程的影响。

4.1 生成测试集

首先，通过实验比较 IFQ 调度器和 MFAP 调度器的性能。实验包括 3 个用户，用户 1 连续发出请求，用户 2 和 3 间隔发出请求。用户 2 在检测到用户 1 累计发出 20 MB 的读请求后，连续发出 10 MB 的读请求；用户 3 每秒发出 10 个均匀时间间隔的读请求。所有请求都是对存储设备的随机访问，单次请求访问存储的数据粒度为 512 KB。我们的启发式算法划分时间片段的粒度为 100 秒。IFQ 调度器实际与理论的分配权重如图 5(a)所示。可以发现，IFQ 调度器很好地保证了干扰公平。与基于干扰公平应得权重相比，MFAP 调度器实际分配权重的误差明显增大，如图 5(b)所示。

4.2 真实测试集

应用冷启动是典型的存储资源延迟敏感使用场景。在冷启动过程中，应用需要交替地使用存储资源和计算资源。我们选取 4 个被广泛使用的应用：LibreOffice Impress，Mozilla Firefox，Google Chrome 和 GNU Image Manipulation Program (GIMP)。将这些应用分别与 4 个吞吐率密集的存储访问应用混合，并对比 CFQ 调度器、STF 调度器[25]、MFAP 调度器、IFQ 调度器与简单提升延迟敏感应用优先

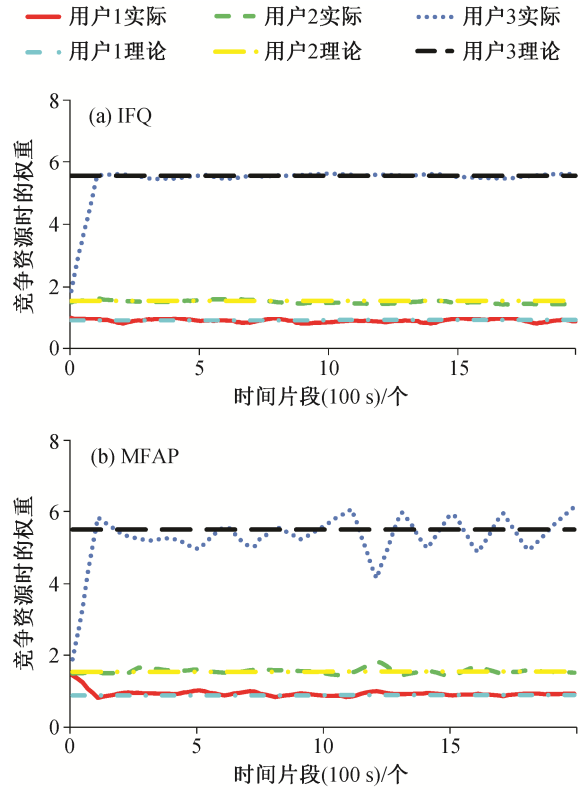


图 5 IFQ 和 MFAP 下用户分配的实际权重和基于干扰公平应得的理论权重对比

Fig. 5 Comparison between the user's assigned weight and the due weight based on interference fairness under MFAP and IFQ

级的CFQ-P调度器的性能，如图6所示。这里我们使用归一化性能，即以单独执行为基准，值越大表示性能越好。可以看到，应用冷启动时间在基于时间片轮转的公平CFQ调度器下的性能最差；由于考虑延迟敏感应用的权重性能，STF调度器在冷启动的性能比CFQ有所提升；IFQ和MFAP应用冷启动的

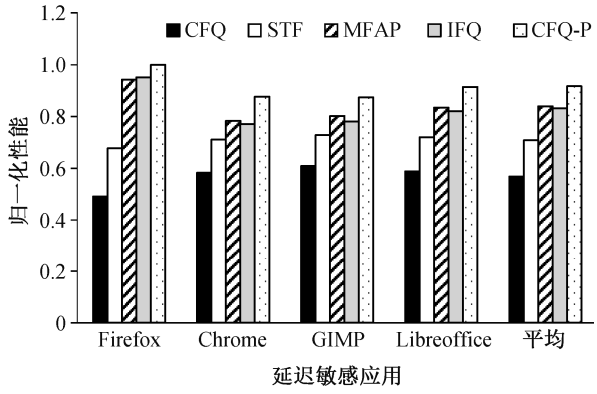


图 6 不同调度器下归一化的应用冷启动时间

Fig. 6 Normalized application cold start time under different schedulers

性能非常接近，并进一步提升，且保证了很好的公平性；CFQ-P调度器在应用冷启动的性能最好，但是简单粗暴地提升应用优先级的方法破坏了公平性。因此，可以认为IFQ保证了与MFAP一致的公平性能。

4.3 真实访问踪迹模拟

选取来自UMass[24]的交互式服务的4个I/O访问踪迹(WebSearch1，WebSearch2，Financial1和Financial2)进行评测。这些访问踪迹都是延迟敏感的，我们只选取其中的存储读操作。使用一个模拟器模拟访问踪迹的行为，并将其与6个吞吐率密集的存储访问应用混合。对UMass访问踪迹的平均访问延迟的评测结果如图7所示，IFQ仍然保证了与MFAP一致的公平性能。

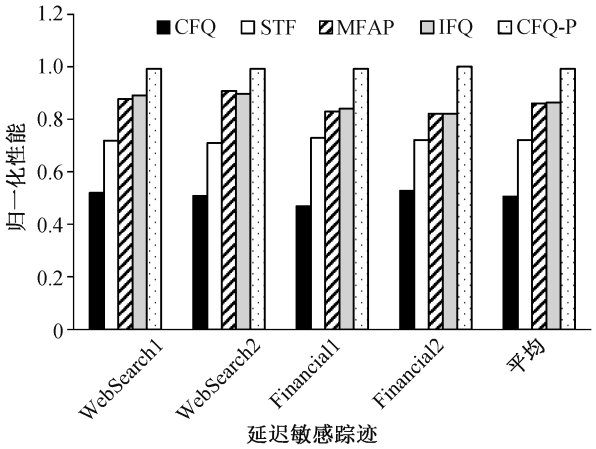
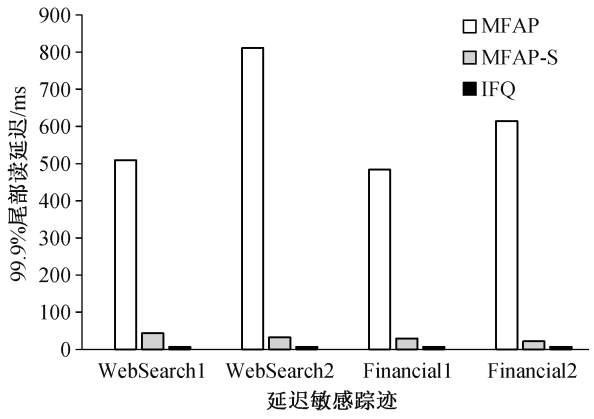


图 7 归一化的平均访问延迟

Fig. 7 Normalized response delay under different schedulers



MFAP-S 代表短时间片 MFAP (将基本时间片尺寸从 100 ms 缩小至 10 ms)

图 8 在 MFAP，MFAP-S 和 IFQ 调度器下 99.9%的尾部延迟

Fig. 8 99.9% tail delay under MFAP, MFAP-S and IFQ

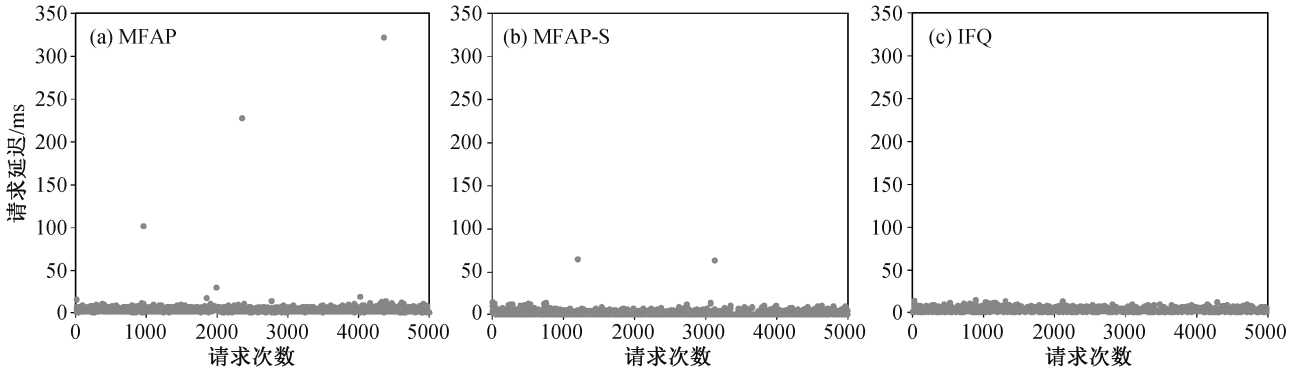


图 9　3 个调度器的请求延迟分布结果

Fig. 9　Three scheduler requests delay distribution results

4.4 访问延迟对比

对比 MFAP 与 IFQ 的访问延迟可以发现，从平均结果来看，两种算法几乎没有区别，但在延迟敏感应用访问存储延迟的长尾问题上，IFQ 算法有明显的性能优势。

依然使用 UMass 的 4 个访问踪迹进行测试，将访问踪迹模拟器与 6 个吞吐率密集读应用混合，并测试读延迟的分布。最差情况的读延迟(99.9%的尾部延迟)结果如图 8 所示。可以看出，IFQ 有效降低了最差情况的读延迟。

通过一个只包含读操作的 Apache 测试，对比 MFAP，MFAP-S 和 IFQ 的请求延迟分布情况(图 9)，结果显示，IFQ 方法可以有效地将读延迟分布压缩在一个较小的范围。

5 总结

本文开发一种新的存储 I/O 调度器 IFQ，在 SSD 设备下，可以在保证干扰公平的同时保证存储 I/O 的延迟。IFQ 的设计动机来源于干扰公平的 MFAP 调度器的高延迟问题和 SSD 设备的特性，包括并行访问、随机访问和性能不稳定等。基于这些特点，我们使用应用访问存储的数据量来评估应用消耗的资源，并基于 SFQ(D)公平队列算法实现 IFQ 调度器。未来，可以将本文方法扩展到其他可并行访问的资源调度以及在分布式存储系统环境下应用的资源分配等。

参考文献

[1] Mao H, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters // Proceedings of the ACM Special Interest Group on Data Communication. Beijing, 2019: 270–288

[2] Mitra S, Mondal S S, Sheoran N, et al. DeepPlace: learning to place applications in multi-tenant clusters // Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems. Hangzhou, 2019: 61–68

[3] Zhang G, Lu R, Wu W. Multi-resource fair allocation for cloud federation // 2019 IEEE 21st International Conference on High Performance Computing and Communications. Zhangjiajie, 2019: 2189–2194

[4] 王金海，黄传河，王晶，等. 异构云计算体系结构及其多资源联合公平分配策略. 计算机研究与发展，2015, 52(6): 1288–1302

[5] Caprita B, Chan W C, Nieh J, et al. Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems // USENIX Annual Technical Conference. Marriott, 2005: 337–352

[6] Agrawala A K, Bryant R M. Models of memory scheduling. ACM SIGOPS Operating Systems Review, 1975, 9(5): 217–222

[7] Demers A, Keshav S, Shenker S. Analysis and simulation of a fair queueing algorithm. ACM SIGCOMM Computer Communication Review. 1989, 19(4): 1–12

[8] 鞠海玲，崔莉，黄长城. EasiCC: 一种保证带宽公平性的传感器网络拥塞控制机制. 计算机研究与发展，2008, 45(1): 16–25

[9] Wong T M, Golding R A, Lin C, et al. Zygaria: storage performance as a managed resource // 12th IEEE Real-Time and Embedded Technology and Applications Symposium. San Jose, 2006: 125–134

[10] Ghodsi A, Zaharia M, Hindman B, et al. Dominant

resource fairness: fair allocation of multiple resource types // Proceedings of the 8th USENIX conference on Networked systems design and implementation. Boston, 2011: 323–336

[11] Elnably A, Du K, Varman P. Reward scheduling for QoS in cloud applications // 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Ottawa, 2012: 98–106

[12] Elnably A, Wang H, Gulati A, et al. Efficient QoS for multi-tiered storage systems // USENIX Conference on Hot Topics in Storage & File Systems. Boston, 2012: 1–5

[13] Dolev D, Feitelson D G, Halpern J Y, et al. No justified complaints: on fair sharing of multiple resources // Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. Cambridge, 2012: 68–75

[14] Wang H, Varman P. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation // Proceedings of the 12th USENIX Conference on File and Storage Technologies. Santa Clara, 2014: 229–242

[15] Gutman A, Nisan N. Fair allocation without trade // Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2. Valencia, 2012: 719–728

[16] Sui Yan, Yang Chun, Tong Dong, et al. MFAP: fair allocation between fully backlogged and non-fully backlogged applications // 2016 IEEE 34th International Conference on Computer Design. Phoenix, 2016: 576–583

[17] Goyal P, Vin H M, Cheng H C. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. IEEE/ACM Transactions on Networking, 1997, 5(5): 690–704

[18] Shen Kai, Park S. FlashFQ: a fair queueing I/O scheduler for flash-based SSDs // 2013 USENIX Annual Technical Conference. San Jose, 2013: 67–78

[19] Mahajan K, Singhvi A, Balasubramanian A, et al. Themis: fair and efficient GPU cluster scheduling for machine learning workloads [EB/OL]. (2019–10–29) [2019–11–10]. https://arxiv.org/abs/1907.01484

[20] Gu J C, Chowdhury M, Shin K G, et al. Tiresias: A GPU cluster manager for distributed deep learning // 16th USENIX Symposium on Networked Systems Design and Implementation. Boston, 2019: 485–500

[21] Axboe J. Linux block IO — present and future // Proceedings of Linux Symposium. Ottawa, 2004: 51–61

[22] Haque M E, He Yuxiong, Elnikety S, et al. Few-to-many: incremental parallelism for reducing tail latency in interactive services. ACM SIGARCH Computer Architecture News, 2015, 43(1): 161–175

[23] Liu Haikun, He Bingsheng. Reciprocal resource fairness: towards cooperative multiple-resource fair sharing in IaaS clouds // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, 2014: 970–981

[24] Storage Performance Council. UMass trace repository [EB/OL]. (2019–07–17) [2019–10–01]. http://traces.cs.umass.edu/index.php/Storage

[25] Mutlu O, Moscibroda T. Stall-time fair memory access scheduling for chip multiprocessors // Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. Chicago, 2007: 146–160