Multi-Queue Fair Queuing

论文背景综述

姓名：祁旋　　学号：M201973010

1. 本文背景

本篇文章出自 USENIX Annual Technical Conference 的 2019 年的会议上，是由美国罗切斯特大学的 Mohammad Hedayati，Michael L. Scott 以及谷歌的 Kai Shen，Mike Marty 发表，USENIX Annual Technical Conference 是一个在 1975 建立的致力于 Unix 及类似系统研究和开发的“Unix Users Group”，并逐渐发展为一个高级计算机系统组织，主要致力于计算机操作系统方面的实践，开发和研究。是操作系统、体系结构方面最好的会议之一，每年召开一次，代表着计算机系统结构方面的顶级会议。类似的会议还有：

- OSDI - Usenix Operating Systems Design and Implementation：在 citeseer 排名中，最好的会议是 OSDI，这是一个收录范围相当广的会议。
- SOSP - ACM Symposium on Operating Systems Principles：两年开一次，和 OSDI 轮流开，由于这两个会议方向很广，因此影响很大。
- FAST - File and storage system：方向比较专一，但水平相当高，是文件和存储最好的会议之一。
- ASPLOS - Architectural Support for Programming Languages and Operating Systems：系统、构架、编程语言三个领域交叉的最好会议。

而且在 2018 年 3 月 21 日，美国计算机协会(ACM)宣布，将 2017 年图灵奖授予 John L. Hennessy 和 David A. Patterson，以表彰他们开创了一种系统的、定量的方法来设计和评价计算机体系结构，并对 RISC 微处理器行业产生了持久的影响。他们设计更快、更低功耗和精简指令集计算机(RISC)微处理器创建了一个系统化的量化方法。这足以显示在现代的计算机系统中，对于计算机的构架优化还有很大的进步空间。除了对硬件的不断更新外，对于整个计算机系统调度的优化也尤为重要。

而本文就是在现代设备的处理速度越来越快的背景下提出的，对于 IO、网

络、计算的设备可以执行每秒百万次的 IO，所以对于现代的多核处理器来说，必须针对多个硬件线程进行独立的 IO 调度，而传统的多队列的实现方式是在每个核心上设置一个 IO 调度队列，但每个队列之间没有缓存一致性流量，即每个队列之间没有交互，这使得整个 IO 队列产生不公平的缺陷，而如果设置一个多队列一致流量，又会使得跨多队列的开销非常大，因为每秒百万次的 IO 之间的时间间隔与处理器中断的时间相当，在多核之间进行序列化请求是不现实的。另一种极端的做法是删除调度程序来达到全带宽，这又会破坏操作系统的公平性和性能隔离。对此，文章提出了 Multi-Queue Fair Queuing（多队列公平排队），用来解决在尽可能少的影响带宽的情况下实现队列的公平性与可伸缩性。

2. 主要思想

为了克服队列间的不公平问题，以伸缩的方式容纳具有内部并行性的多队列设备，确保每个流接受它所共享的带宽，在队列中引入一个阈值 T，当某两个队列的调度窗口差值大于 T 时，先暂停较快的队列的调度，以便调度较慢的队列赶上，当慢速队列赶上进度后，就可利用中断通知阻塞队列继续执行。这样就可以在不与其他核心通信的条件下进行全局的公平调度，只要所有的队列的请求的时间差在阈值 T 的范围内，各个队列间就可并行的分发请求，这也使得当 T=0 时，与传统的无多队列通信的情况一致。

而在队列的中断请求中。利用了一种新的数据结构：令牌树来实现本地化同步，其原理为：这多个队列之间由来自同一个 thread、core、socket 组成一棵令牌树，记录了每个队列节点的距离，当每次 IO 完成释放插槽时，优先给本地的队列重用，其次令牌树的结构允许快速的分配到最近的一个队列， 这样可最小化的同步开销。

由于为了达到设备的全带宽，必须一直保持设备的繁忙，这就需要在前一个请求完成之前发送一个新的请求，与此同时，调度机制要决定请求的顺序，这就要避免发送比实际并行个数更多的请求，保留对请求的调度排序能力，在此之上，作者引入了参数 D，来表示所有队列中允许的未分配请求的最大数量。

由于 MQFQ 是基于 Start Fair Queueing 来实现公平排队的，它沿用了 SFQ 中定义的 Start flag 和 Virtual time，并做了小改进，作为一个请求的优先顺序。

3. 验证评估

在排队队列中，最重要的两个属性是公平性与高效。所以这两方面的验证必不可少。在实验的平台上，选用了 linux 的操作系统，评估了 MQFQ 在两个快速多队列设备上的公平性和性能，包括 RDMA 上的 NVME 和多队列的 SSD，测试结果为在 SSD 固态上能够实现 0.5M IOP/S，而在更高速的 NVME 上实现了 4M IOP/S。则表明 MQFQ 能够达到较快的请求调度。

首先，在公平性上，通过公式的推导得出了当上述阈值 T=0 时，MQFQ 能够提供与 SFQ 相同的公平性，在整个算法中表示了公平性和可伸缩性的权衡，而在实际的验证过程中需要根据具体的环境设置不同的参数，虽然单线程工作负载可以承受 T = 0，但是当跨多个 socket 的多个线程提交较小的请求时就需要更加大的 T 值，为此，当确立了最大堆积请求数 D 时，为满足实现最大的吞吐量，需要提供较大的 T 值。并将 MQFQ 与现有系统上的两种调度方法进行比较：1）NOSHCED：适用于快速并行设备的无 IO 调度算法，是无争用的；2）BFQ：Linux4.12 以后操作系统上使用的 IO 调度算法，它只有一个中央调度器。在这三中调度方式上进行了三种测试：

- FIO：具有良好的可伸缩性，能够灵活的配置 IO 模式，它对每个 IO 请求的处理较少，可生成已知特征的负载来与其他进程抢占资源。
- FLASHX：一种图形处理框架，利用 SSD 扩展大型数据集，它的 IO 请求较少。
- AEROSPIKE：一个 flash 优化的键值存储，为保持高性能，直接对原始设备进行 IO 读写，能够达到较频繁的 IO。

实验结果是在带宽的占用方面,NOSHCED 能够达到最大的吞吐量，这得益于无调度程序，其次，MQFQ 能够达到 NOSHCED 的 70%左右的吞吐量，而 BFQ 的吞吐量极其小，可以看出在加入了多队列的调控后，在速度这一项上，MQFQ 能够获得较好的处理速度，而 BFQ 无法满足快速的 IO 请求，单一的中央调度局限较大。而在上述三种测试方面，在 FLASHX 与 FIO 的联合争用资源上，MQFQ

能够在低于两倍（认为可接受范围）的速度降低情况下得到较好的性能，而 BFQ 和 NOSHCED 无法同时在 SSD 和 NVME 上保持两倍以内的减速。在 AEROSPIKE 和 FIO 的资源竞争上，MQFQ 基本上能保证两倍以下的减速，而 BFQ 缺乏对并行调度的支持，它导致 FIO 的运行速度降低了近 15 倍。

实验结果表明在无论是在伸缩性还是在公平性上，MQFQ 均能实现较好的性能平衡，能够满足现今多核多线程对队列调度的要求。

4. 总结

在整片文章中，作者从现今多队列无法满足公平性的要求开始，考虑了对同步开销和可伸缩性的均衡，提出了 MQFQ 的多队列公平排队调度模式，解决了高速 IO 对调度器来不及调度的压力问题。通过利用令牌树来本地化同步减小了开销，通过设置两个参数来提供可伸缩性，能够针对不同的系统设置最佳的配置。在较小的影响吞吐的情况下达到了较好的调度公平。