高技术通讯 2014 年 第24 卷 第6 期:587 ~596

doi:10.3772/j.issn.1002-0470.2014.06.006

基于优先级的时限感知的数据中心网络拥塞控制算法①

赵正伟②[*,**]　许　刚[***]　毕经平[*]

([*]中国科学院计算技术研究所网络技术研究中心　北京 100190)

([**]中国科学院大学　北京 100049)

([***]山东科技大学信息科学与工程学院　青岛 266590)

摘　要　分析了数据中心的应用特性、流量特征以及目前 TCP 协议存在的不足,指出目前数据中心的大规模交互式网络应用具有软实时性,因而受时限约束,而网络应用的划分-聚合设计模型及采用的公平共享的拥塞控制协议,是导致网络流错过时限的主要原因。据此,提出了一种基于优先级的时限感知的数据中心传输控制协议(PD^{2}TCP),一种新的网络拥塞控制算法。该算法在交换机端,根据瞬时队列长度和单一门限进行显式拥塞通告(ECN)标记;在主机端,根据流的时效性需求及其历史信息,赋予其不同的优先级,并根据流的优先级和网络的拥塞程度调整拥塞窗口。同时在小规模的真实环境中和大规模的仿真环境中对 PD^{2}TCP 的性能进行评价。实验表明,与时限感知的数据中心 TCP(D^{2}TCP)相比,PD^{2}TCP 错过时限流的比例降低了 65%,流完成时间的 99th分位数降低了 45%,并且几乎没有降低延迟不敏感的背景流的吞吐率。PD^{2}TCP 能够和 TCP 共存,因而可以在真实环境中部署。

关键词　数据中心网络,时限,优先级,拥塞控制,显式拥塞通告(ECN)

0　引　言

近年来,随着互联网和云计算的快速发展,数据中心已经成为许多在线服务的关键基础设施,如 Web 搜索、在线零售、广告/推荐系统以及社交网络等。这些服务通常是软实时性的交互式应用,有严格的时限(deadline)约束,错过时限会导致用户访问和运营收入明显减少[1-3]。然而,由于这些服务采用的划分-聚合(partition-aggregate)设计模式以及目前数据中心普遍采用的公平共享(fair sharing)的传输层协议,目前仍然有很大一部分(7% ~25%)流错过时限[4]。针对这一问题,最近的研究工作分别提出了针对数据中心网络的拥塞控制算法和流调度算法,以及在网络中引入时限感知(deadline-aware)来减小网络延迟,满足流的时限约束。

数据中心传输控制协议(data center TCP, DCTCP)[5]是一个基于显式拥塞通告(explicit congestion notification,ECN)的拥塞控制算法,它根据网络的拥塞程度巧妙地调节拥塞窗口。但是,DCTCP 是公平共享(fair sharing)的协议,并且是时限不可知的,造成超过 25% 的流错过时限[6]。D^3[4]率先在数据中心中引入时限感知来满足流的时限,交换机根据流的大小和时限来分配带宽。然而由于 D^3 采用集中式的调度和主动式的速率请求,虽然与 DCTCP 相比有了性能提高,但是它仍有严重的性能和实用问题[6]。PDQ[7]是一个抢占式的分布式流调度算法,能够近似实现 EDF 和 SJF 调度策略,与其他工作相比性能有大幅提升[8],但需要在交换机上维护流的状态以及交换机之间的同步与协作,因而通信和计算开销太大,不适合大规模的数据中心网络环境。时限感知的 DCTCP(deadline-aware data-center TCP, D^{2}TCP)[6]在 DCTCP 算法中引入了时限感知,根据网络拥塞程度和流的时限来调节拥塞窗口。但是,D^{2}TCP 没有明确区分延迟敏感的流和延迟不敏感的流之间的优先级,可能造成在短暂的拥

① 973 计划(2011302505)和国家自然科学基金(60803138/F0208, 61070210, 61303243)资助项目。

② 男,1982 年生,博士生;研究方向:网络测量与监控,数据中心网络,下一代互联网;联系人,E-mail: zhaozhengwei@ict.ac.cn

(收稿日期:2013-09-18)

塞期间,延迟敏感流的分组可能排在延迟不敏感流的分组后面,这增加了延迟敏感流错过时限的可能性。针对这种情况,本文提出了一种基于优先级的时限感知的数据中心传输控制协议(priority-based deadline-aware datacenter TCP, PD^{2}TCP)算法,其主要目标是减小错过时限的流的比例,同时不伤害背景流量的吞吐率。PD^{2}TCP 继承了 TCP 的分布式和反应式(reactive)的特性,并且明确区分了延迟敏感的流和延迟不敏感的流之间的优先级关系。在网络拥塞时,根据网络拥塞程度和流的优先级不同程度地退避拥塞窗口。本文还用锯齿模型(sawtooth model)分析了 PD^{2}TCP 的稳定状态行为,并解释了其性能提高的原因。小规模的真实实现和大规模的仿真实验都表明,与已有算法相比,PD^{2}TCP 的性能有显著提高。

1 背景和相关工作

1.1 数据中心的应用特性

划分-聚合。划分-聚合设计模式是目前许多大规模的、面向用户的 Web 应用的基础。如图 1 所示,来自用户的请求被逐层分解并指派给底层的工作节点(workers)计算完成后,把它的计算结果汇聚到其父节点,最终在根节点形成完整的响应并返回给用户。Web 搜索、社会网络、广告推荐系统,以及 MapReduce 和 Dryad 等数据处理服务都遵从该设计模式。

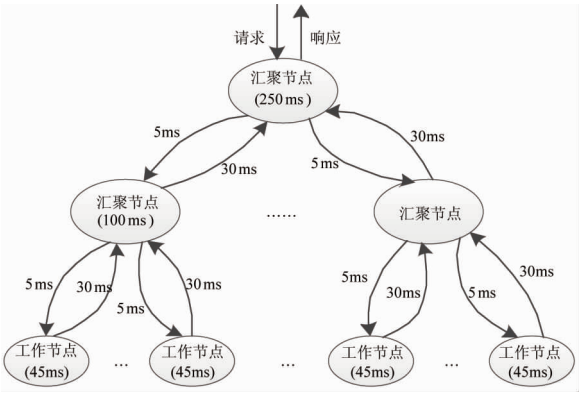

图 1 划分-聚合设计模式

应用的时限。Web 应用的交互性决定了延迟是影响用户体验的关键因素。文献[9]指出,除了 Internet 和渲染(rendering)延迟,典型的应用大概有 200 ~ 300ms 的服务水平协议(service level agreement,SLA)来完成操作并向用户返回响应[10,11]。这个 SLA 分解到图 1 中的各个层次,如最底层的工作节点须在 45ms 内完成计算,并在 30ms 内完成传输。时限期满,根汇聚节点就把已有的结果返回给用户,错过了时限的工作节点将不会对最终结果做出任何贡献,这不仅导致响应质量降低,还浪费了计算资源和网络带宽。

这里的时限包含两部分:计算时间和通信延迟。这两者是相互矛盾的:预留给通信更多的时间可以降低网络错过时限的比例,但是这也意味着留给计算的时间就少了,因而可能降低最终结果的质量(如在 Web 搜索中,用于 Page Rank 的时间更少了)。理想情况下,不但希望降低错过时限的流的比例,同时能尽量减小留给通信的时间,从而为计算留出更多的时间。

目前数据中心中,对于延迟敏感的流,错过时限是一种普遍现象。如文献[4]所述,即使在宽松的时限约束下(40ms),超过 7% 的流错过时限。

1.2 数据中心的负载特征

数据中心的负载大概可以分为两类:延迟敏感的流量和吞吐率敏感的流量,有些延迟敏感的流量(划分-聚合设计模式产生的流)还具有高同步突发性。

延迟敏感性。面向用户的在线数据密集型应用的流量,如 Web 搜索、在线购物和广告/推荐系统等,以及协调集群运行的状态更新和短消息,是延迟敏感的流量。这类流量有软实时性,如果延迟超过特定阈值,会降低用户体验或者造成集群状态不一致。因此我们为这些流设定时限,度量其性能的主要指标是满足时限的流的比例,有的文献也称其为应用吞吐率(application throughput)[4,7]。

吞吐率敏感性。数据中心中有些后台流量是吞吐率敏感的,如大量数据的更新操作、倒排索引表的更新、镜像文件的操作等。这些流量用于提取和组织大规模的数据,以保持响应的质量。它们对延迟不敏感,只要求吞吐率越高越好。对这类流,我们用吞吐率作为其主要的性能度量指标。

同步突发性。划分-聚合设计模式的应用产生的流,不仅是延迟敏感的,同时更重要的是具有同步突发性(synchronized burstiness)[4,5,12]。如图 2 所示,由于工作节点几乎同时完成计算并开始传送结果,导致在很短的时间内,很多流同时汇聚到交换机的同一个接口上,将会耗尽交换机的内存或者该接口允许的最大缓冲,导致丢包。即使包很小,这种现象也会发生。目前数据中心的交换机普遍采用共享

的浅缓冲架构,缓冲极易被大流耗完,因而不能吸收这些同步突发,造成丢包,错过时限。该现象也被称为 Incast 问题[12-14],或者扇入突发(fan-in burst)[6]。

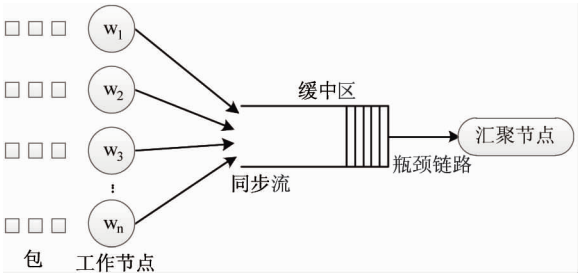

图 2　同步突发

流的大小和并发性。文献[5]对一个生产用数据中心的测量研究表明:延迟敏感的流,大小一般小于 1MB,吞吐率敏感的流一般大于 1MB。同步突发流的并发性由划分-聚合设计模式决定,其中位数大概是一个机柜中的服务器数目;吞吐率敏感的流并发性较小,中位数是 1,75th分位数是 2。

总之,在数据中心中,吞吐率敏感的流,延迟敏感的流和高突发性的流同时并存。

1.3　数据中心的 TCP

同步突发是数据中心流量的一个基本特性,但是目前的数据中心网络不能很好地处理这种情况,导致应用错过时限,具体如下:

(1)TCP/IP 协议栈在网络发生拥塞时,使用丢包作为对发送端的反馈。在这种机制下,发送端必须等待一段时间来检测到分组的丢失,即使时限已经截止。检测到丢包后,发送者降低一半的发送速率来减缓拥塞,造成发生丢包的流错过时限。目前数据中心解决这个问题通常同时采用两种方法:(a)提高链路带宽;(b)增加用于通信的时间,例如使其大于延迟的 99th分位数。前者将增加成本,后者减少了节点的计算时间,导致响应质量降低,或者导致需要更多的机器来弥补每台机器减小的计算时间,继而加剧了扇入突发,因为扇入度增大了。

(2)TCP 平等对待所有的拥塞流,在最小化流的完成时间以及错过时限流的比例方面,远不是最优的。最佳情况是,网络优先对待即将错过时限的流。但是,TCP 是一个公平共享的协议,缺乏基于时限的区分,造成延迟敏感的流错过时限;并且 TCP 感知不到时限,因此并不十分适合数据中心[4]。

更严重的是,由于 TCP 不能很好地满足应用的时限要求,开发人员转而寻求其他解决途径。例如,据报道,Facebook 正在开发基于 UDP 的拥塞控制协议,来满足应用的延迟要求[5,15,16]。

1.4　相关工作

关于拥塞控制、网络调度和减小延迟的文献非常多,这里只列出与本文最相关的研究工作。

流调度算法: Hedera[17]周期性地对大流进行重新映射,DevoFlow[18]采用集中式的控制器对大流进行调度。两者的调度周期都在秒级,不能满足数据中心的低延迟要求,因为通常数据中心中流的时限约束在数十个毫秒。

拥塞控制算法及时限感知:D^3[4]率先把时限感知引入数据中心,但是它采用集中式的主动调度策略,其性能严重依赖流的达到次序。除此之外,D^{3}不能与 TCP 共存,严重影响其在真实环境中的部署。DCTCP[5]根据网络的拥塞程度而不是拥塞的发生来调节发送速率,能够同时满足数据中心的高突发、低延迟和高吞吐率的要求。DCTCP 是公平共享的,平等对待所有流,不考虑不同流的时效性需求。D^{2}TCP[6]把时限感知引入到 DCTCP 中,并提出优先对待离时限截止近的流。但是,它没有清楚地指明延迟敏感的流和延迟不敏感的流之间的优先级划分,可能导致延迟敏感的流排在延迟不敏感流之后,增加了延迟敏感的流的平均完成时间,甚至导致错过时限。HULL[19]通过影子队列(phantom queue)提前发出拥塞信号,并结合 DCTCP 算法自适应地控制拥塞,网络延迟能够接近交换结构(fabric)的固有延迟并同时获得很高的带宽利用率。但它是公平共享的,并且是时限不可知的。

多路径传输机制: DeTail[20]是一个网络侧的多路径感知的拥塞管理机制,旨在减小流完成时间的长尾分布,而不是以减小平均的流完成时间或者是减小错过时限流的比例为目的。MPTCP[21]在一条 TCP 连接中开启多条子流(subflow),并且在拥塞时在这多条子流之间进行负载均衡。对于大于 70KB 的流,MPTCP 非常有效;但是对短流,在拥塞时,它没有足够的时间进行重新调度,因此不能很好地满足小流的时限。

2　PD^{2}TCP 算法

2.1　算法概述

算法设计约束和假设:(1)不修改交换机的软硬件,仅要求其支持显式拥塞通告(ECN);(2)算法能够和目前常用的 TCP 协议共存;(3)所有的改动

仅限在主机端;(4)流在建立时已知其大小和时限。

算法的主要目标:提高满足时限的流的比例,但不能以牺牲吞吐率敏感流的性能为代价。同时,网络能较好地容忍由划分-聚合设计模式带来的扇入突发。

算法的基本思想包括优先级思想和分布式与反应式(reactive)思想。

(1)优先级的思想。延迟敏感流的优先级高于延迟不敏感的流;离时限截止近的流的优先级高于离时限截止远的流的优先级。在网络拥塞程度相同的情况下,优先级高的流的拥塞窗口退避幅度小于优先级低的流,这就相当于后者为前者腾出,使其能够尽早完成。

(2)分布式和反应式的思想。分布式算法能够支持大规模的数据中心,并且容易扩展,同时不需要更改交换机的软硬件。由于可能存在多条优先级相当的流争用带宽,造成网络短暂的过度订阅(over-subscription),PD^{2}TCP 的反应式方式能够根据后续的拥塞信息被动地调节,保证这种过度订阅是暂时的和小范围的。

2.2 基于优先级的分布式调度

集中式的调度算法,如 EDF 和 SJF,依赖全局的流信息,对大规模的数据中心来说这在技术上是不可行的,因为延迟敏感的小流到达速率很高,基本上不可能实时获取这些流的全部信息。除此之外,集中式算法也存在单点失效、中心控制器的开销过高以及扩展性差等问题。所以数据中心中的任何流调度机制,都必须在不完整信息的情况下做出决策,分布式调度更适合数据中心。

此外,由于缺乏优先级机制,目前的数据中心很难同时满足延迟敏感的流和延迟不敏感的流的需求。数据中心是一个共享的网络环境,其中的流有不同的时效性需求,发生扇入突发导致的拥塞时,延迟敏感流的分组可能会排在延迟不敏感流的分组之后,增大了前者错过时限的可能性。因此有必要采取基于优先级的调度机制,满足不同流的需求;不区分流的相对优先级关系,对延迟敏感的小流是很不公平的。

2.3 算法实现

2.3.1 网络的拥塞程度

和 DCTCP[5] 一样,我们采用 ECN 机制来获取网络的拥塞程度。在交换机上采用简单的标记机制,只要队列长度超过一个很小的阈值,就设置 Congestion Experience 代码点,否则不设置。这是为了保证当队列超过阈值时,发送端能很快得到通知。在实现上只要将交换机 RED 机制的 low 和 high 门限都设置为,并按照瞬时队列长度标记即可。

接收端按照 CE 标记的顺序把拥塞信息传递回发送端。最简单的方法就是每个分组一个 ACK,只要 CE 码点被设置,ECN-ECHO 标记就被设置。这与 RFC1368 有所不同,对于延迟确认,可参考文献[5]。

发送端根据被标记分组的比例,调节拥塞窗口。这里的关键点是发送端从一系列只有一位的标记信息中,获取了多位的反馈。可以从这些反馈中获取网络的拥塞信息。

发送端维护一个变量,用来估计已标记分组的比例,每发送完一个窗口的数据(大概是一个 RTT 时间)更新一次:

$$\alpha = (1 - g) \times \alpha + g \times F \tag{1}$$

式(1)中 F 表示最后一个窗口中被标记的分组的比例。$0 < g < 1$,是一个权重,是新采样的比例相对于上一次采样的 α 的权重。由于当交换机队列长度大于 K 时,对每个分组发送端都收到一个标记,当队列长度小于 K 时,发送端接收不到标记,所以式(1)表明:α 就是队列占用大于 K 的概率,α 接近 0 表明拥塞程度较低,而 α 接近 1 表明拥塞程度较高。α 就是网络的拥塞程度。

2.3.2 流的优先级

为了减小在拥塞时延迟敏感的流排在延迟不敏感的流后面的概率,以及离时限截止近的流排在离时限截止远的流后面的概率,我们为流引入了优先级的概念。优先级越高,拥塞时其拥塞窗口退避幅度越小。流的优先级定义如下:

$$p = \frac{R_D}{R_A} \tag{2}$$

其中 R_D 是在时限截止前发送完一条流剩余数据的期望速率,R_A 是从流开始到当前时刻的平均速率。

需注意的是,本文 PD^{2}TCP 与 D^{2}TCP 最大的区别在于引入了完全优先级机制,如图 3 所示。该图展示了在相同的网络拥塞程度下,不同类型流的优先级划分机制。PD^{2}TCP 算法对延迟约束极端严格(小于 5ms)的流赋予最高优先级($p = 2$),因为没有足够的 RTT 来让这些流从容地调节拥塞窗口以满足其时限约束。无时限约束的背景流的优先级最低($p = 0.5$)。其他有时限约束的流的优先级则在最高和最低优先级之间动态变化。D^{2}TCP 没有明确区分无时限约束的流(non-deadline)和离截止期较

远的流(far-deadline)之间的优先级,在拥塞发生时,后者可能为前者让步。除此之外,D^{2}TCP 没有考虑时限约束极端严格的流,并且背景流量的优先级是一个中间值,高于离时限截止较远的流的优先级,这增加了错过时限流的比例,同时也增大了流的平均完成时间。

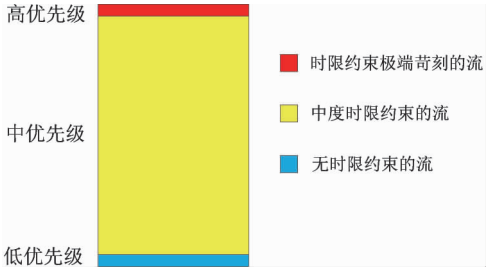

图 3　PD^{2}TCP 的优先级机制

2.3.3　拥塞窗口的调节

为了根据网络的拥塞程度和流的优先级来调节拥塞窗口,我们定义惩罚函数:

$$P = \alpha^p \tag{3}$$

在图像处理领域中,这个函数被称为 Gamma 校正函数。这个函数的一个优良特性是当自变量 α 在[0, 1]之间变化时,其函数值也在这个区间变化,这就可以保证,当拥塞加剧时或者时限即将结束时,PD^{2}TCP 可以像 TCP 一样收敛。

接下来,根据惩罚函数 P 调节拥塞窗口:

$$W = \begin{cases} W \times (1 - \dfrac{P}{2}), & P > 0 \\ W + 1, & \text{for} \quad P = 0 \end{cases} \tag{4}$$

这里,我们根据网络的拥塞程度而不是拥塞的发生来调整拥塞窗口,这是和传统的 TCP 拥塞控制不同的。TCP 的其他特性,如慢启动、加性增和超时处理等都没有改变。

可以看出,在网络拥塞程度相同的情况下,优先级越高,惩罚函数越小,窗口退避得越少,从而保证能够以较快的速度发送完毕。在所有的流中,拥塞程度相同时,延迟不敏感的流窗口退避程度最大,从而为延迟敏感的流腾出带宽,满足后者的时限约束。

2.3.4　提前优先级翻转

当可以确定流不能满足其时限约束时,提前将其优先级降为最低,为其他延迟敏感流腾出带宽。当流满足以下两个条件之一,就执行该操作:

(1)时限已经截止了,但流还没有完成传输;

(2)即使以网卡最大速率发送,到时限截止还不能完成发送。

2.3.5　参数设置

本文根据文献[5]来设置 DCTCP、D^{2}TCP 和 PD^{2}TCP 的关键参数。对 1Gbps 的链路,设置 g = 1/16。对 PD^{2}TCP,和文献[6]一样,本文限制了流的优先级 p 的取值范围在[0.5, 2]之间。对所有的协议,本文都把 RTOmin 设置为 20ms。

2.3.6　算法分析

考虑两条无限长的流共享一条瓶颈链路,优先级分别为 ph(高优先级)和 pl(低优先级)。进一步假设这两条流完全同步,并具有相同的 RTT,这样两者的锯齿窗口(sawtooth)就有相同的震荡周期,如图 4所示。两者窗口的震荡幅度是不同的,其差值可用下面的公式计算:

$$\begin{aligned} \Delta W &= W_{\max}(1 - \frac{\alpha^{ph}}{2}) - W_{\max}(1 - \frac{\alpha^{pl}}{2}) \\ &= W_{\max}\frac{-\alpha^{ph} + \alpha^{pl}}{2} \end{aligned} \tag{5}$$

两条流的平均窗口大小分别为高优先级流的平均窗口大小

$$\begin{aligned} W_{ah} &= \frac{1}{2}(W_{\max} + W_{\max}(1 - \frac{\alpha^{ph}}{2})) \\ &= W_{\max}(1 - \frac{\alpha^{ph}}{4}) \end{aligned} \tag{6}$$

和低优先级流的平均窗口大小

$$\begin{aligned} W_{al} &= \frac{1}{2}(W_{\max} + W_{\max}(1 - \frac{\alpha^{pl}}{2})) \\ &= W_{\max}(1 - \frac{\alpha^{pl}}{4}) \end{aligned} \tag{7}$$

平均窗口大小的相对差值为

$$\delta = \frac{W_{ah} - W_{al}}{W_{ah}} = \frac{\frac{1}{2}\Delta W}{W_{ah}} = \frac{-\alpha^{ph} + \alpha^{pl}}{4 - \alpha^{ph}} \tag{8}$$

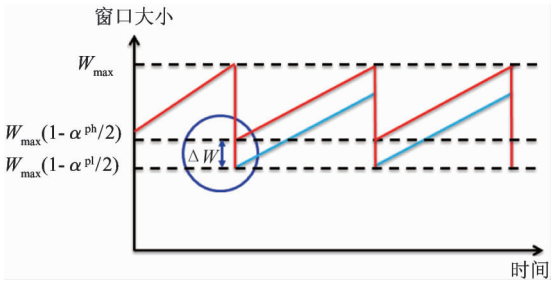

图 4　两条具有不同优先级的同步流的窗口变化情况

因为流的吞吐率与平均窗口大小成正比,所以 δ 就是流的优先级从 ph 降到 pl 时,高优先级流吞吐率的损失。图 5 显示了 α 从 0 变化到 1 时,δ 的变

化曲线。我们把 ph 设置为 1,把 pl 设置为 0.5,分别代表了 D²TCP 和 PD²TCP 中没有时限约束的背景流的优先级。可以看出,当 α 趋近 0.3 时,δ 达到最大值,约 7%。这表明,与 D²TCP 相比,PD²TCP 算法的背景流量损失了不到 7% 的吞吐率。实际上,在我们的实验当中,这个损失的吞吐率都在(0, 4%)区间。背景流损失的这些吞吐率为延迟敏感的流腾出了空间,因而可以减少错过时限的流的比例,以及流的平均完成时间。

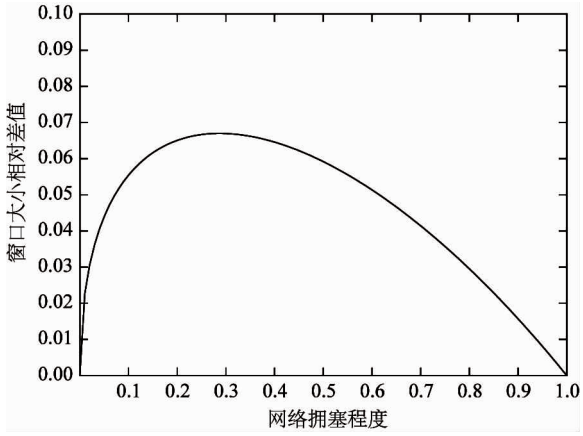

图 5 优先级分别为 1 和 0.5 的两条流的窗口差值

3 实验结果

本文同时使用真实实现和仿真环境来评价 PD²TCP 算法的性能。真实实现主要用来运行小规模的基准测试(micro benchmarks),以及用来验证仿真环境的参数设置。本文用仿真环境来评价在大规模的数据中心中,真实工作负载的情况下,PD²TCP 的性能。本节首先描述了本文采用的流量模型,接着展示了在真实环境中,小规模的基准测试结果,并且验证了仿真环境的参数设置。最后,在更大规模的拓扑下,更复杂的工作负载的情况下,展示了 PD²TCP 的性能。

3.1 流量模型

根据文献[5],本文的流量模型包含 3 种类型的流量:查询、短消息和背景流量,分别代表高同步突发的流、延迟敏感的流和吞吐率敏感的流。它们的到达速率服从 Poisson 分布,大小符合均匀分布,但分布区间不同。为了反映传统的流量突发的性质,我们周期性地调整流的达到速率。流的并发数目也根据文献[5]设置。

3.2 真实环境

3.2.1 环境搭建

按照图 6 所示,搭建了一个实验环境,模拟了数据中心的多层树形拓扑结构。该环境共使用 1 个核心交换机,4 个机柜交换机(Top of Rack, ToR),和 16 台服务器,跨越 4 个机柜。图中每个节点都是一台服务器,配置为 4 核的 Intel Xeon 处理器,内存 4G,操作系统是 Fedora 17,内核版本 3.3。所有的链路都是 1Gbps,并且经测试发现,每台服务器 5 个网卡的情况下可以实现全双工线速转发。为了与目前数据中心交换机的浅缓冲保持一致,我们为交换机的每个网卡设置了 128kB 的缓冲[4]。还有 2 个产生流量的服务器,用于向数据中心发送查询请求。同一个 ToR 交换机下的两台服务器之间的 RTT 大约是 500。

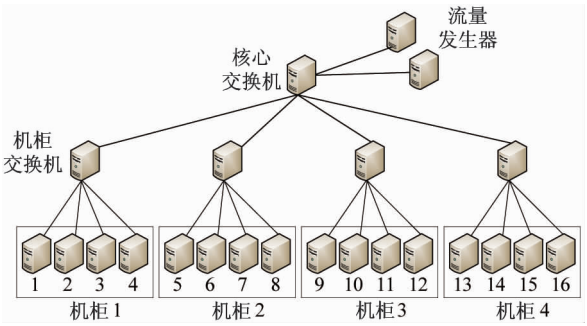

图 6 真实环境拓扑

为了使协议栈感知到流的时限信息,本文扩展了目前通用的 Linux 系统的 socket API,应用程序通过套接字接口把流的大小和时限传递到内核协议栈。

3.2.2 真实环境下的实验结果

为了与 D²TCP 进行比较,并且对模拟器参数设置进行验证,本文同时在真实环境和仿真环境中进行了如下实验:

在每个机柜中,随机选取一台机器作为汇聚节点,其他机器为工作节点。为了实现 20、30 和 40 的扇入度,每台物理机器模拟多个工作节点。由于所有的工作节点几乎同时向汇聚节点发送响应,所以汇聚节点到 ToR 的链路就成了瓶颈。

查询流量:为了模拟 Web 搜索的场景,从流量发生器模拟向汇聚节点发送查询请求。查询请求平均到达速率 1000 次/秒,工作节点的响应流均匀分布在[2kB, 50kB]区间上。时限约束为 30。

短消息:我们随机选取一半的工作节点向另一半的工作节点发送短消息,由此来模拟数据中心工作节点之间的控制状态更新。短消息流的平均达到

速率 300 次/秒,大小均匀分布在[50kB, 1MB]区间上。时限约束为 40ms。

背景流量:在每个机柜中,随机选取 2 台服务器向另外一台发起一条长流,这代表了数据中心背景流并发程度的 75th 分位数。平均达到速率 10 次/秒,大小均匀分布在[1MB , 50MB]的区间上。背景流量没有时限约束。

图 7 显示了在真实实现和仿真环境下,D^{2}TCP 和 PD^{2}TCP 错过时限的流的比例。3 组数据的扇入度分别为 20,30,40。可以看出,在这三种情况下,PD^2-real 错过时限的流的比例都小于 D^2-real,并且随着扇入度数的增大,两者的差距明显增大。扇入度数为 40 时,D^2-real 错过 18% 的时限,而 PD^2-real 则错过 13% 。

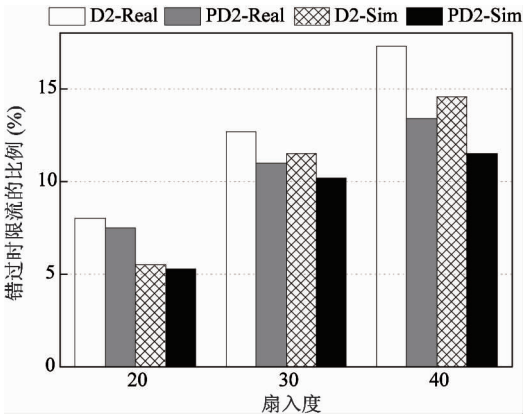

图 7　真实和仿真环境下的小流

图 8 显示了在 3 组扇入度下,没有时限约束的背景流的吞吐率。可以看出,PD^2-real 和 D^{2}TCP-real 的差别不大,相差都在 2.5% 以内。因此可以说,延迟敏感的流性能的提高,不是以牺牲吞吐率敏感的流的性能为代价的。

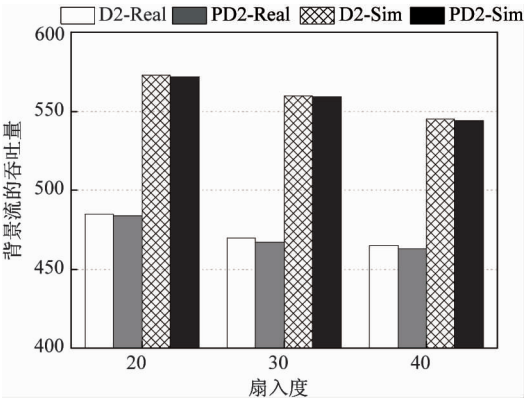

图 8　真实和仿真环境下的背景流

3.2.3　模拟器验证

为了验证仿真环境参数设置,本文比较了图 7 和图 8 中真实环境和仿真实验的结果。可以看出,仿真结果稍稍偏离真实环境结果。例如,对错过时限流的比例,仿真结果要小于真实实现结果,但是对长流的吞吐率,仿真结果要明显高于真实结果。这些差异是由于仿真环境不能捕捉真实系统所有的细节和细微的差异,如不可预测的系统事件、中断的合并以及 Large Segment Offload(LSO)等。

但是,PD^{2}TCP 和 D^{2}TCP 相对的性能差异,以及这种差异的变化趋势,在真实实现和仿真环境下是基本相同的。例如,图 7 中扇入度为 30 时,在仿真环境下,PD^{2}TCP 错过时限流的比例比 D^{2}TCP 减小 13% ,而在真实实现中减小 11% 。扇入度增大时,仿真环境和真实实现的性能差异都是增加的。由于这些关键的相似性,我们相信大规模的仿真结果是可信的。

3.3　仿真实验

我们在 NS2 上实现了 DCTCP, D^3, D^{2}TCP 和 PD^{2}TCP。仿真环境采用典型的数据中心的 Fat-Tree 拓扑结构,如图 9 所示。实验中一共有 25 个机柜,每个机柜 40 台服务器,一共 1000 台。每台服务器通过 1Gbps 的链路与 ToR 交换机相连。由于数据中心网络的瓶颈通常在 ToR[12,5,4],我们把核心层和汇聚层交换机抽象为一个理想的交换机。ToR 到理想交换机的链路速率是(一个机柜中服务器的数量)。我们把 ToR 交换机的缓冲设置为 4MB[5],由所有的端口共享。链路延迟设置为 20,平均的 RTT 为 500。

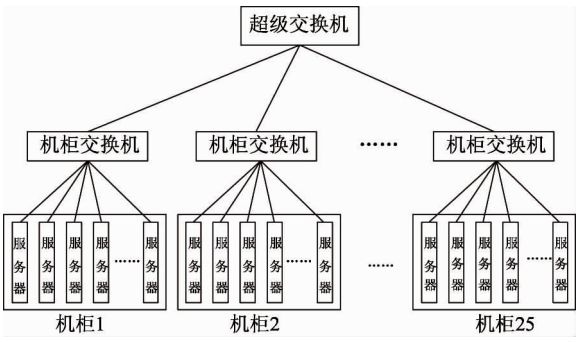

图 9　仿真环境拓扑

仿真实验中使用的流量模型基本和真实环境中一样,除了流达到速率。为了反映传统的网络流的突发特性,本文周期性地调整流到达速率。对查询流,本文以 100 为周期,首先让查询请求以2500 次/秒持

续一个突发时间，然后再以 1000 次/秒持续到这个周期结束。突发时间持续 20 ~ 50。对短消息，到达速率在 100 次/秒到 500 次/秒之间变化。对长的背景流，其并发数目在 1 ~ 4 之间变化，中位数是 1，75th分位数为 2。

时限的设置：这里的时限由两部分组成：基准时限和一个随机的变化量。基准时限分别是 20、30 和 40。变化量分为低、中、高 3 个级别，其中前两个表示给基准时限增加一个均匀的随机量，分布区间分别为[0，10%]和[0，50%]，第三个则表示时限分布服从指数分布，均值为基准时限。

图 10、11、12 分别是低度、中度以及高度变化情况下错过时限流的比例。轴表示错过时限的流的比例，轴表示扇入度数，反映流的同步并发程度。目前数据中心中，多数的在线数据密集型应用的扇入度都在 5 ~ 40 之间[4]。

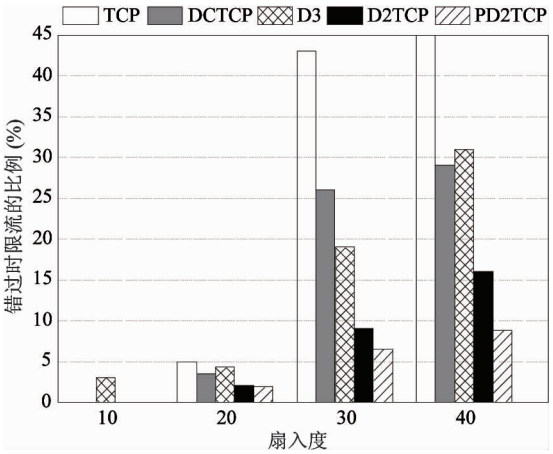

图 10　时限低度变化时错过时限情况

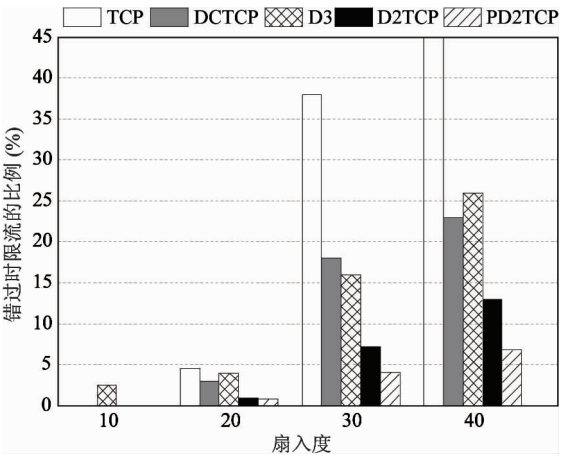

图 11　时限中度变化时错过时限情况

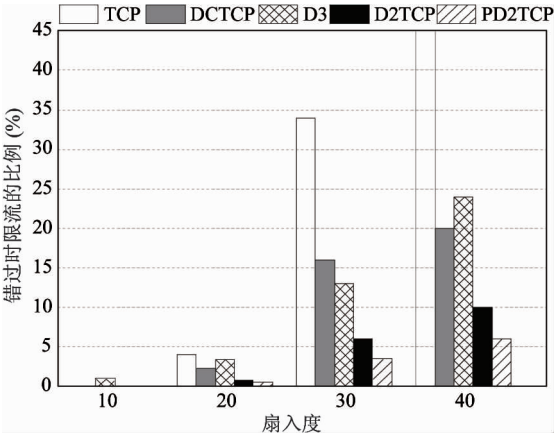

图 12　时限高度变化时错过时限情况

在这 3 个图中，TCP 错过时限的比例都随着扇入度的增加而迅速增加。虽然与 TCP 相比，DCTCP 和 D^3 有大幅减小，但是它们仍然有很大一部分流错过时限。例如，在时限中度变化的情况下，扇入度为 40 时，DCTCP 和 D^3 错过了 13% 的流的时限。相比之下，在扇入度为 40 时，D^{2}TCP 错过时限的流的比例一直维持在 9% 以下。而 PD^{2}TCP 约错过了 4% 的流的时限。由于在 3 种时限变化情况下都有这种趋势，我们相信 PD^{2}TCP 是稳定的，足以应对一系列的时限分布。

5 种算法在时限变化更大的情况下都表现得更好，这是因为更大的时限变化平滑掉了扇入突发带来的拥塞延迟。

为了解释上述结果，大家来看一组数据。图 13 显示了在时限中度变化情况下，DCTCP、D^3，D^{2}TCP 和 PD^{2}TCP 延迟的 50th，90th，99th分位数，图中的纵坐标是延迟，并且被归一化到时限。每条线上的 3 个点分别表示延迟的 50th，90th，99th分位数，可以看出，PD^{2}TCP 比 D^{2}TCP 有明显的降低。

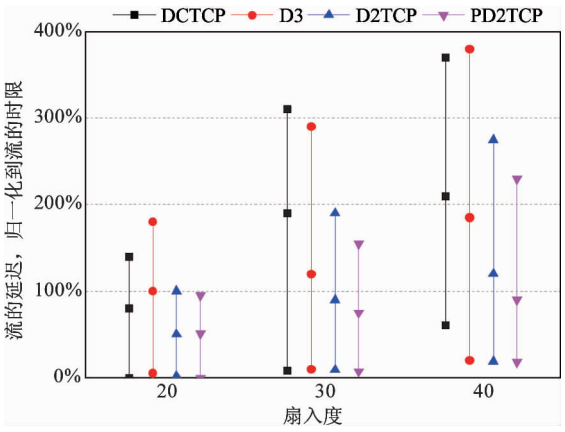

图 13　延迟的 50th，90th，99th分位数

图 14 显示了在不同的扇入度下,吞吐率敏感流的性能。图中带宽被归一化到 TCP 的吞吐率上。可以看出,4 个算法的性能都不错,至少能到达 TCP 性能的 87%。DCTCP, D^2TCP 和 PD^2TCP 比 TCP 表现要好,PD^2TCP 至少能获得和 TCP 一样的吞吐率。但是要比 D^2TCP 的吞吐率略小,因为 PD^2TCP 降低了背景流量的优先级。

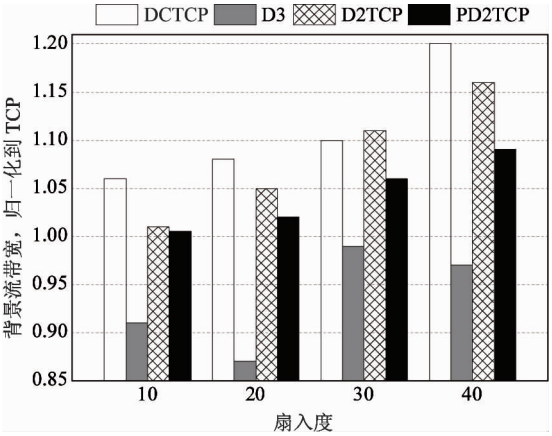

图 14　背景流量的吞吐率

4　结 论

数据中心应用的交互性决定了延迟是影响用户体验的关键因素之一,这意味着这些应用是有时限约束的。然而此类应用通常采用基于树的算法,会形成同步的扇入突发,在交换机上造成短暂的拥塞,导致丢包和重传,错过应用的时限。同时,缺少优先级的拥塞控制算法也不能很好地满足应用的时限。本文提出了一种基于优先级的时限感知的数据中心网络拥塞控制算法,旨在满足数据中心应用的时限约束。通过赋予延迟敏感的流较高的优先级,在网络拥塞时,其拥塞窗口退避幅度较小,从而满足其时限约束。真实实现和仿真环境都表明 PD^2TCP 能大幅提高满足时限的流的比例,同时几乎没有伤害吞吐率敏感的流的性能。此外,PD^2TCP 能够和 TCP 共存,因而可以在真实的数据中心中部署。

参考文献

[1] Hoff T. Latency is Everywhere and it Costs You Sales - How to Crush it. http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html. 2009

[2] Kohavi R, Longbotham R. Online experiments: Lessons learned. *Computer*, 2007, 40(9): 103-105

[3] Brutlag J. Speed matters for Google web search, http://services.google.com/fh/files/blogs/google_delayexp.pdf, Google, 2009

[4] Wilson C, Ballani H, Karagiannis T, et al. Better never than late: Meeting deadlines in datacenter networks. In: Proceedings of the ACM SIGCOMM 2011 Conference, New York, USA, 2011. 50-61

[5] Alizadeh M, Greenberg A G, Maltz D A, et al. Data center TCP (DCTCP). In: Proceedings of the ACM SIGCOMM, New York, USA, 2010. 63-74

[6] Balajee Vamanan, Hasan J, Vijaykumar T N. Deadline-aware datacenter TCP (D2TCP). In: Proceedings of the ACM SIGCOMM, New York, USA, 2012. 115-226

[7] Hong C Y, Caesar M, Godfrey P B. Finishing flows quickly with preemptive scheduling. In: Proceedings of the ACM SIGCOMM, New York, USA, 2012. 127-138

[8] Alizadeh M, Yang S, Sharif M, et al. Pfabric: minimal near-optimal datacenter transport. In: Proceedings of the ACM SIGCOMM, New York, USA, 2013. 435-446

[9] Kohavi R, Longbotham R, Sommerfield D, et al. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 2009, 18(1):140-181

[10] Kohavi R, Henne R M, Sommerfield D. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In: Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining, New York, USA, 2007. 959-967

[11] DeCandia G. Dynamo: amazon's highly available key-value store. In: Proceedings of twenty-first ACM symposium on Operating systems principles, New York, USA, 2007. 205-220

[12] Chen Y P, Griffith R, Liu J D, et al. Understanding TCP incast throughput collapse in datacenter networks. In: Proceedings of the 1st ACM workshop on Research on enterprise networking, New York, USA, 2009. 73-82

[13] Amar P. Measurement and analysis oftcp throughput collapse in cluster-based storage systems. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies, Berkeley, USA, 2008. 12:1-14

[14] Vasudevan V, Phanishayee A, Shah H, et al. Safe and effective fine-grained TCP retransmissions for datacenter communication. In: Proceedings of the ACM SIGCOMM, New York, USA, 2009. 303-314

[15] Rothschild J. High performance at massive scale: Lessons learned at facebook. mms://video-jsoe.ucsd.edu/calit2/JeffRothchildsFacebook.wav. 2009

[16] Saab P. Scaling memcached at Facebook, http://www.

facebook. com/note. php? note _ id = 39391378919. 2008

[17] M Al-Fares, Radhakrishnan S, Raghavan B, et al. Hedera: Dynamic flow scheduling for data center networks. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, Berkeley, USA, 2010. 281-295

[18] Curtis A R, Jeffrey C M, Jean T, et al. DevoFlow: Scaling flow management for high-performance networks. In: Proceedings of the ACM SIGCOMM, New York, USA, 2011. 254-265

[19] Alizadeh M, Kabbani A, Edsall T, et al. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, Berkeley, USA, 2012. 19-32

[20] Zats D, Das T, Katz R H. DeTail: Reducing the flow completion time tail in datacenter networks. In: Proceedings of the ACM SIGCOMM, New York, USA, 2012. 139-150

[21] Raiciu C, Barre S, Pluntke C, et al. Wisscik D. M. Handley. Improving datacenter performance and robustness with multipath tcp. In: Proceedings of the ACM SIGCOMM, New York, USA, 2011. 266-277

A priority-based deadline-aware congestion control algorithm for datacenter networks

Zhao Zhengwei[*,**], Xu Gang[***], Bi Jingping[*]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] College of Information and Engineering, Shandong University of Science and Technology, Qingdao 266590)

Abstract

The characteristics of datacenter applications as well as their traffic flows and the the inefficiency of state-of-the-art transmission control protocol (TCP) are analyzed, and it is pointed out that the large-scale interactive web applications in today's datacenters have soft-real-time natures so there are deadline constraints associated with network flows, and that misseddeadlines are mainly caused by the partition-aggregate design pattern of the applications, combined with the fair-sharing congestion control protocols. Based on the analysis, the Priority-based Deadline-aware Datacenter TCP (PD^{2}TCP), a novel congestion control algorithm is proposed. PD^{2}TCP assigns different priorities to flows according to their timeliness requirements and history information, and modulates the congestion window based on the priorities and the extent of congestion. The performance of the PD^{2}TCP was evaluated in a small-scale real environment and a large-scale environment, and the results showed that, compared to the deadline-aware datacenter TCP (D^{2}TCP), the PD^{2}TCP reduced the fraction of deadline-missing flows and the 99th percentile normalized flow completion time by 65% and 45%, respectively. The PD^{2}TCP can co-exist with TCP, thus can be deployed in real datacenters.

Key words: datacenter networks, deadline, priority, congestion control, explicit congestion modification (ECN)