QoS 队列调度算法研究及应用

李典斌　刘星沙　夏明伟

中南大学铁道学院网络中心　湖南　410075

摘要：本文概述了常用队列调度算法的实现机制，并举例说明了在特定的网络环境中如何应用 QoS 队列调度算法。

关键词：服务质量；队列调度；通用处理器共享；加权公平队列

0 引言

队列调度算法是实现网络 QoS 控制的核心机制之一，是网络资源管理的重要内容，通过控制不同类型的分组对链路带宽的使用，使不同的数据流得到不同等级的服务。通常调度算法的工作模式可以分为两种：工作保留模式和非工作保留模式。如果队列中有数据包等待发送服务器就工作的调度算法称为工作保留调度算法；如果队列中有数据包等待发送但服务器仍然处于空闲状态的调度算法称为非工作保留调度算法。例如，即使服务器处于空闲状态，同时队列中有数据包等待发送，但是为了等待下一个高优先级的数据包，服务器也会推迟当前数据包的传输，这种调度算法就属于非工作保留调度算法。当数据包的传输时间很短时，非工作保留调度算法几乎是不公平的。

调度算法的另一种分类方法是根据调度算法的内部结构来划分的，主要有两种：基于优先级分类的调度算法和基于帧结构的调度算法。在基于优先级的调度算法中有一个称为虚拟时间的全局变量。调度算法根据该变量为每个数据包计算一个时间戳，然后根据时间戳对数据包排序和调度。虚拟时钟，加权公平队列都属于这种结构。基于优先级的调度算法的实现复杂度取决于两个因素：更新优先级列表算法和选择最高优先级数据包算法的复杂度(至少是O(log V)，其中V是共享输出链路的队列数)和计算时间戳算法的复杂度(这主要取决于所采用的调度算法，加权公平队列(WFQ)的时间戳的计算复杂度为O(V)，虚拟时钟的计算复杂度只为O(1))。

在基于帧结构的调度算法中，时间被分为固定长度或可变长度的帧。每个数据流所能使用的带宽资源就是每一帧中所允许传输业务量的最大值。存储转发队列是帧长度固定的基于帧结构的调度算法，在这种结构中，如果一帧中数据流的业务量小于预留带宽，服务器就会空闲。加权循环队列，差额循环队列允许帧长度可变，同时，如果一个数据流的业务量小于预留带宽时，下一个数据流就可以提前被调度。基于帧结构的调度算法最大的优点是实现简单，成本低，最大的缺点是缺乏灵活性和扩展性。

1 典型的调度算法简介

1.1 先进先出队列(FIFO)

FIFO 队列是最简单的基于优先级的调度算法。在 FIFO 队列中数据包的时间戳就是数据包的到达时间。FIFO 队列提供了基本的存储转发功能，也是目前因特网中使用最广泛的一种方式，它采用默认的排队方法，不需要配置。其优点是实现简单，成本低，缺点是不能提供 QoS 功能和隔离技术，缺乏公平性，易于受到非法用户的攻击。

1.2 严格优先级调度算法(PQ)

严格优先级调度算法维护一个优先级递减的队列系列并且只有当更高优先级的所有队列为空时才服务低优先级的队列(如图 1 所示)。假设队列 1 比队列 2 具有更高的优先权，队列 2 比队列 3 具有更高的优先权等等。只要链路能够传输分组，队列 1 尽可能快地被服务。只有当队列 1 为空，调度器才考虑队列 2。当队列 2 有分组等待传输且队列 1 为空时，队列 2 以链路速率接受类似地服务。当队列 1 和队列 2 为空时，队列 3 以链路速率接收服务等等。

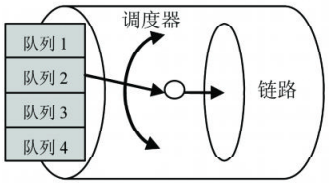

图 1　严格优先权调度器

然而该调度机制会使低优先级队列处于饥饿状态。例如，如果影射到队列 1 的数据流在一段时间内以 100% 的输出链路速率到达，调度器将从不为队列 2、3、4 服务。避免队列饥饿需要上游路由器精心规定数据流的业务特性以确保映射到队列 1 的业务类不超出输出链路容量的一定比例，这样可以使队列 1 常常为空，允许调度器为低优先级队列服务。

严格优先级调度算法对低时延业务非常有用。假定数据流 X 在每一个节点都被映射到最高优先级队列，那么当数据流 X 的分组到达时，如果调度器是空闲的，则分组被立即服务。

1.3 通用处理器共享算法(GPS)和加权公平队列算法(WFQ)

1.3.1 通用处理器共享

作者简介：刘星沙(1956-)，女，高级工程师，研究方向：计算机网络及应用。

GPS 算法是一种理想的调度算法(如图2所示)，是根据流模型定义的，也就是假设数据包是可以被无限细分的。在GPS算法中，假设服务器的处理速率恒为r。在t时刻，如果数据流i的数据包在队列中等待处理，就认为数据流i在t时刻处于激活状态。假设有正整数Φ_1，Φ_2，Φ_3，Φ_4代表各数据流的权重，$S(\tau,t)$是时间间隔(τ,t)中服务器为数据流i提供的服务，那么对于时间间隔(τ,t)内任何暂存在服务器中的数据流i获得的服务$S_i(\tau,t)$，GPS 算法定义为$\frac{S_i(\tau,t)}{S_j(\tau,t)}\geq\frac{\Phi_i}{\Phi_j}$，j=1,2…N。

假设r为服务器的处理速率，将数据流j累加可得$S_i(\tau,t)\sum_j\Phi_j\geq(t-\tau)r\Phi_i$。根据轮换对称性，任意数据流i的保证速率(最小服务速率)为$g_i=\frac{\Phi_i}{\sum_{\forall j}\Phi_j}r$。

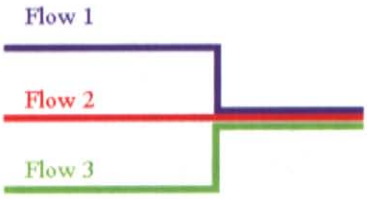

图2　通用处理器共享

GPS 算法具有如下特性：

(1) 假设是数据流i的平均速率，只要$r_i\leqslant g_i$，就可以保证数据流i独立于其他数据流的吞吐率p_i。还可以保证数据流i的瞬时数据以大于等于g_i的速率被处理。

(2) 由于数据流i在任意时刻获得的服务独立于其他数据流，数据流i的时延抖动是自己队列长度和达到时间的函数，独立于其他连接的队列长度和到达时间。其他调度算法如FIFO和PQ没有这种性质。

(3) 通过改变Φ_i我们可以不同的方式处理不同的数据流。例如，当所有的Φ_i都相同时，GPS就退化为均衡处理器共享。另外只要数据流的平均速率之和小于，不论怎么分配Φ_i，系统总是稳定的。

(4) 通过增加Φ_i就可以减少数据包经历的时延，虽然这种方式是以牺牲其他数据包p的时延作为代价的，但是当激活数据流稳定时，这种代价并不是很大。所以GPS算法和速率整形器联合使用时可以得到性能优良的调度器，为数据流提供最坏情况下的时延和时延抖动保证。

1.3.2 加权公平队列算法(WFQ)

GPS 算法最大的缺点是不能处理长度可变的数据包。WFQ算法是GPS算法的近似。假设F_p是数据包P在GPS算法中的离开时间，那么WFQ算法就是一个模拟GPS算法并按升序调度数据包的工作保留算法，也就是说WFQ算法总是选择F_p值最小的数据包进行调度。下面介绍WFQ算法的虚拟时间实现。

假设时间t_j是事件j^{th}的发生时间(同时发生的事件可以任意排序)，服务器的处理速率为r。在服务器中，第一个发生事件的时间记为$t_1=0$。可以看出在时间间隔(t_{j-1},t_j)内处于激活状态的数据流是固定的，我们将它记为集合B_j。当服务器空闲时，虚拟时间$V(t)$记为0，那么WFQ的$V(t)$计算如下：

$$\begin{cases}V(0)=0\\V(t_{j-1}+t)=V(t_{j-1})+\frac{t}{\sum_{i\in B_j}\Phi}r,t\leq t_{j-1},j=2,3\ldots\end{cases}$$

$V(t)$的变化率为$\frac{\partial V(t_j+\tau)}{\partial\tau}$为$\frac{1}{\sum_{i\in B_j}\Phi_i}r$，每个暂存的数据流接收到的处理速度为$\Phi_i\frac{\partial V(t_j+\tau)}{\partial\tau}$。假设数据流$i$的第$k$个数据包的到达时间为$a_i^k$，长度为$L_i^k$，$S_i^k$和$F_i^k$分别表示这个数据包的开始虚拟时间和结束虚拟时间，$F_i^0=0$，那么，我们可以得到：

$$\begin{cases}S_i^k=mak\{F_i^{k-1},V(a_i^k)\}\\F_i^k=S_i^k+\frac{L_i^k}{\Phi_i}\end{cases}$$

从实现的角度来看，WFQ算法的虚拟时间实现有两个重要优点：

(1) 数据包的完成时间决定于数据包的到达时间和上一个数据包的完成时间；

(2) 数据包根据完成时间升序被处理。

WFQ使用虚拟时间存在的缺点：跟踪集合B_j需要花费很大的开销。

1.4 虚拟时钟算法(Virtual Clock)

虚拟时钟算法根据数据包的到达时间和用户定义的保留速率计算数据包的时间戳。假设TB_i^k是数据流i的第k个数据包的时间戳，p_i是数据流i的保留速率，AT是长度为L_i^k的数据包的实际到达时间，那么数据包的时间戳定义如下：

$$TS_i^k\longleftarrow\max(AT,TS_i^{k-1})+\frac{L_i^k}{\rho_i}$$

如果数据包比预期的到达时间晚，那么经过最大延迟L_i^k/ρ_i后数据包被传输；如果数据包比预期的到达时间早，在最坏情况下，数据包被传输的时间为$TS_i^{k-1}+L_i^k/\rho_i$。最坏情况下的服务质量不受其他连接行为的影响。

1.5 存储转发队列(Stop-and-Go)

存储转发队列将输入和输出链路的时间轴分为固定长度的时隙“帧”。在两帧之间到达的数据包只能在下一帧中被传输，在同一帧中的数据包可以任何次序传输，因此每个数据包都被引入一个固定的时延θ。如果数据包的最大到达速率小于帧中保留的时间片，那么这种算法能确保有限的时延和时延抖动。存储转发队列有两个问题：因为算法是非工作保留的，所以没有静态统计复用收益；时延θ与帧分配的颗粒度有关，选择小的帧长度可以获得小的θ，但是，为了得到好的带宽利用率应该选择大一些的帧长度。

1.6 循环队列(round-robin)

循环队列通过循环服务避免局部队列饥饿。调度器总是顺序地移到下一个有分组要发送的队列(空队列被跳过)。如果每个队列都有分组等待发送，调度顺序和队列顺序匹配；如果一些队列为空，则其它队列被频繁地服务。在极端情况下，如果其它队列都为空，单个队列就可以使用全部链路带宽。

当分组进入一个空队列时，该队列在下一个循环中被服务，这样就可以避免队列“饥饿”。循环调度的缺点是分组时延难于改进，它不可能为低时延业务分配专用队列。每个队列的服务间隔完全依赖于那段时间内其他队列中有多少分组等待发送以及这些分组的长度，这些变量难以准确预测，所以RR调度容易产生时延抖动。调度器可以通过改变服务顺序(例如采用顺序1，2，3，2，4，2，1，2，…)更频繁地调度某些队列以给这些队列更频繁的传送机会，然而分组大小的随机分布仍然会造成时延抖动问题。

1.7 差额循环队列(DRR)和加权循环队列(WRR)

DRR算法是RR算法的扩展。DRR算法为每个队列分配一个常量Q_N(以权重为比例的时间片)和一个变量D_N(差额)。Q_N反应了该队列可以发送长期平均字节数。D_N的初始值为零且当队列为空时复位为0。当DRR算法服务一个新队列时，调度器复位计数器B_{sent}(表示该循环已经从队列中发送的字节数)。当下面两个条件满足时，DRR算法从队列中发送分组：(1)队列中有分组等待发送；(2)(Q_N+D_N)大于等于(B_{sent}+队列中下一个分组的长度)。否则，该队列的差额D_{N+1}被置为Q_N+D_N－B_{sent}，调度器按顺序移到下一个队列。Q_N+D_N表示在服务时间间隔内队列能够发送最大字节数，在一定程度上D_N可以平滑数据流的突发。队列通过Q_N可以获得长期的相对带宽分配。如果激活队列的数目小于N，则激活队列可以根据Q_N值共享未用的输出链路带宽。

WRR算法非常类似于DRR算法。WRR算法采用类似的时间片和差额的概念，但是算法稍有不同。在WRR中，当队列发送(B_{sent})的字节数超过队列允许的限制时(仍为Q_N+D_N)，才对下一个队列进行服务。因此，差额是一个负数值(超出Q_N+D_N的数量)且被当作下一个循环该队列发送的字节数的减少量。

2 应用举例

在某省政府机关的包括省、市(州)、区(县)的三级专线网络中，需要承载视频会议、IP电话及数据等应用，从省到市(州)、市(州)到区(县)的广域网带宽均为2Mbps。为了保证视频会议和IP电话的效果，我们采用严格优先级调度算法(PQ)，分别在广域网边界路由器上对视频会议和IP电话所需要的带宽进行了设置，其中视频会议应用分配了768kbps的带宽，IP电话采用了G.729方式的音频编码，每门电话分配了20kbps的带宽，这样很好的保证了视频会议和IP电话的实时性效果。

3 结束语

为了平衡影响调度算法设计的各种因素，获得比较好的性价比，调度算法的设计必须根据每类业务的特点选择不同的算法。网络的发展趋势是业务多样化，在实际网络中，彼此有分层关系的业务流常常共享链路，这时单一的调度算法无法满足链路带宽共享的需求，需要考虑使用综合结构的调度算法。也就是说，在追求简单性和易实现性的同时，还必须考虑算法的综合性能。

参考文献

[1]Irman Hermadi.Comparing Family of Fair Queuing Scheduling Algorithms for Multimedia Operating Systems.2005.

[2]Abhay K.Parekh Member IEEE and Rober G.Gallager.Fellow.IEEE.A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks:The Single-Node Case.2005.

[上接36页]

progress_id 入侵日志文件的标志，值为0，表示为系统进程，值为1，表示为应用进程；

(4) 将进程按照入侵和非入侵分类，获得入侵进程的函数：HANDLE Result_progress(HANDLE progress_id,int flag, long mem_num,int m,int n,long x,long r) 。

其算法的流程图如图4所示：系数A的取值一般为2或者3，本课题在设计考虑到系统进程在运行的实际情况，A的取值为2。系数A的取值一般为2或者3，本课题在设计考虑到系统进程在运行的实际情况，A的取值为2。

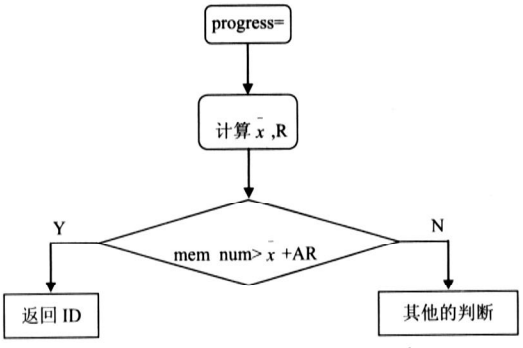

图4 判断进程入侵内存流程图

5 结束语

本课题设计一个具有实时分析数据的系统，通过进程对内存的占用情况、进程占用CPU资源以及入侵端口的分析，从而判别出入侵进程。通过在实验证明，该算法实现了数据特征的分析，比较准确的分析出入侵行为。

参考文献

[1]Lance Spitzner. Honeypots: Definitions and Value of Honeypots. http://www.spitzner.net/.2002.

[2]熊华等.网络安全——取证与蜜罐.人民邮电出版社.2005.

[3]刘云生,李国徽.实时数据库系统中的嵌套事务.软件学报.1999.

[4]Honeynet Project:Know Your Enemy:Honeynets[EB/OL].2006.2.