一种具有 $O(\log N)$ 信息复杂度的高速 crossbar 调度算法

彭来献, 田 畅, 赵文栋
(解放军理工大学通信工程学院, 江苏南京 210007)

摘 要: 本文提出一种可扩展性强的高速 crossbar 调度算法——*i*RGRR(iterative request-grant-based round-robin), 它通过简化处理流程和减小调度开销, 克服了传统算法(例如 *i*SLIP[1]、PIM[2])可扩展性差的缺陷. *i*RGRR 将控制信息复杂度从 $O(N)$ 级大大减小到 $O(\log N)$ 级, 具有良好的可扩展性, 可应用于太比特交换机/路由器中. 仿真结果表明, 在各种不同的均匀和非均匀业务流下, *i*RGRR 能够获得与 *i*SLIP 几乎相同的性能. 另外, *i*RGRR 比 *i*SLIP 具有更好的公平性以及更加易于用硬件实现.

关键词: 路由器; 输入排队; crossbar; 控制信息复杂度; *i*RGRR; 可扩展性
中图分类号: TP393.05 文献标识码: A 文章编号: 0372-2112 (2006) 11-2024-06

A New Scheduling Algorithm with O(logN) Control Messages Complexity for High-Speed Crossbars

PENG Lai-xian, TIAN Chang, ZHAO Wen-dong
(*Institute of Communications Engineering, PLA University of Science and Technology, Nanjing, Jiangsu 210007, China*)

Abstract: This paper presents a scalable scheduling scheme, called iterative request-grant-based round-robin (iRGRR), for high-speed crossbars. It overcomes limitation that most traditional scheduling schemes (such as iSLIP, PIM) suffer from poor scalability by simplifying the execution process and reducing the scheduling overhead. iRGRR dramatically reduces the complexity of control messages from an order of O(N) to O(logN), and thus provides fine scalability, and can be used in terabit switches/routers. The simulation results show that iRGRR can achieve nearly the same performance as iSLIP under various traffic models, including uniform and non-uniform destination distributions. In addition, iRGRR provides better fairness and has lower implementation complexity than iSLIP.

Key words: router; input-queueing; crossbar; control messages complexity; iterative request-grant-based round-robin; scalability

1 引言

随着 Internet 规模和容量的飞速增长, 人们对于具有高速、大容量、可扩展性强的交换机/路由器等网络设备的需求也越来越迫切. 为提高数据的传输效率, 定长交换技术和输入排队(IQ, input queuing) crossbar 交换网络在高速交换机/路由器中得到广泛的应用. 不定长度的数据包在输入端被分割成固定长度的"信元", 经过查表、报头处理以及调度, 由 crossbar 传送到输出端, 再组装成数据包发送到输出链路上. 处理一个信元的时间通常被称为一个时隙. 数据缓存策略和调度算法是影响交换网络可扩展性和性能的两个关键因素. 为避免 HOL 阻塞(Head of Line Blocking)造成的交换网络的最大吞吐量下降为 58.6% 的现象[3], 一般采用虚拟输出排队技术(VOQ, virtual output queuing)来消除 HOL 阻塞, 即一个输入端为每一个输出端维护一个 FIFO 队列. 这样, 信元就会存在输入/输出端竞争: (1)输入端竞争是指, 同一个输入端内到达不同输出端的多个信元, 在同一时隙都需要传输; (2)输出端竞争是指, 位于不同输入端内的多个信元, 在同一时隙都需要到达同一个输出端. 因此, 需要调度算法来解决信元输入/输出端竞争, 建立多条输入/输出端连接, 并控制 crossbar 工作, 保证无冲突地传输信元. IQ crossbar 调度算法通常又被称为二部图匹配算法. 人们已提出众多调度算法[1,2,4~7], 这些算法在可扩展性和实现难易程度上差别很大, 但大部分应用于高速交换网络的调度算法都采用多次迭代的"请求-许可-接受"(*i*RGA, iterative request-grant-accept)处理流程. 一些研究成果已被应用于商用产品中, 例如 PIM 算法被用于 DEC 的 AN2 交换

收稿日期: 2005-09-14; 修回日期: 2006-05-22

机中[2], *i*SLIP 算法被用于 cisco 12000 系列路由器[1].

基于 *i*RGA 的调度算法在实现时, 每个输入(输出)端都有一个独立的仲裁器, 通常所有仲裁器都集中在一个调度器中, 每个输入端控制器(IPC, input port controller)在每个时隙与调度器交互队列状态等控制信息, 以保证算法正确执行. 这些控制信息在高速串行链路上传输. 在本文中, 控制信息量指在一次迭代中, 一个 IPC 与调度器之间交换的信息比特数目. 在 RGA 执行流程中, 每个输入端与调度器至少需要交互 N 比特请求信号和 log*N* 比特接受信号. 考虑一个链路速率为 10Gbps 的交换网络, 一个 64Bytes 长度的信元传输时间为 51.2ns, 也就是说, 控制信息传输时间与调度器执行时间之和必须限定在 51.2ns 内. 根据当前 CMOS 技术, 一个仲裁器只需要 2ns~3ns 的执行时间[8]. 而当前成熟应用的高速串行链路的速率只能达到 2.5Gbps, 即使一个时隙全部用来传输控制信息, 最多也只能传输 128 比特, 如果端口速率增加或调度算法包含多次迭代, $O(N)$ 级的信息复杂度将大大限制交换网络所能支持的端口数目. 因此, 控制信息的传输时延成为基于 *i*RGA 的调度算法可扩展性的瓶颈, 一般只能支持较少的端口数目($N \leqslant 32$)[1,2].

为解决可扩展性问题, 本文提出一种新的高速调度算法 *i*RGRR(iterative request-grant based round-robin), 它的基本思想建立在 *i*SLIP 基础之上, 同时也是对 DRRM[6,7] 算法的改进和升级. 在解决信元输入/输出端竞争时, *i*RGRR 算法使用与 *i*SLIP 算法相同的 round-robin 仲裁方法, 但将处理流程从三个步骤(RGA)简化成两个(RG), 并且将控制信息量的复杂度由 $O(N)$ 级降低到 $O(\log N)$. 另外, *i*RGRR 算法一次迭代的执行结果与 DRRM 算法完全相同, 在均匀业务流下都能获得 100% 吞吐量[6]. 然而, 当 *i*RGRR 算法使用多次迭代时, 在均匀业务流下, 比 DRRM 算法具有明显的时延性能优势; 在非均匀业务流下, 能够提供更好的时延和吞吐量性能. 仿真研究结果表明, 与 *i*SLIP 算法相比, *i*RGRR 算法在不降低性能的同时克服了可扩展性瓶颈, 更加适用于高速、大容量、多端口的交换机/路由器(例如太比特交换机/路由器).

2　传统的基于 *i*RGA 的调度算法

在传统的基于 *i*RGA 的 crossbar 调度算法实现中, 每个 IPC 需要与集中式调度器交换控制信息. 对于一个 $N \times N$ 规模的 crossbar 交换网络, 调度器主要包含 N 个输入端和 N 个输出端仲裁器, 分别对应各个输入/输出端. 算法每次迭代按照"请求-许可-接受"三个步骤依次执行:

Step1: 请求(request) 如果一个未匹配的输入端有信元等待发送, 根据要求发送的输出端口号, 分别向各相关输出端发送请求.

Step2: 许可(grant) 如果一个未匹配的输出端接收到多个请求, 按照仲裁规则从中选择一个, 并向发出该请求的输入端发送许可信号.(解决输出端竞争)

Step3: 接受(accept) 如果输入端接收到多个许可, 按照仲裁规则从中选择一个, 并向发出该许可的输出端发送接受信号, 这样就建立一个匹配边(或称为一个连接).(解决输入端竞争)

各种算法的区别在于输入/输出端仲裁器使用的选择方式不同, 例如 PIM[2] 算法是随机选择. 然而随机选择会导致不公平, 出现"饿死"现象[1], 而且高速随机数产生器会增加仲裁器实现的复杂性. *i*SLIP[1] 和 FIRM[5] 算法都采用 round-robin 选择方式, 这种仲裁方法不仅简单、硬件易实现, 而且克服了不公平的缺陷. FIRM 比 *i*SLIP 具有更好的公平性[5].

在每次迭代中, IPC 需要与调度器交互如下控制信息: (1)输入端 IPC 向调度器至少发送 N 比特的请求信息; (2)调度器执行完毕, 需要回送至少 log*N* 比特的信息, 通知输入端哪一个请求被接受. 如果仲裁器采用其他选择方式, 例如 VOQ 队列长度[4], 将会交互更多的控制信息.

3　*i*RGRR 算法

根据上文的分析, 减少控制信息的交互是提高系统性能和可扩展性的关键. 为解决这个问题, 我们提出一种新的算法 *i*RGRR. 它一次执行过程包含多次迭代, 每次迭代只有"请求-许可"两个步骤, 将控制信息复杂度从 $O(N)$ 级降低为 $O(\log N)$. 同 *i*SLIP 算法一样, 在 *i*RGRR 算法中, 每个输入/输出端也对应一个 round-robin 仲裁器, 用于解决输入/输出端竞争. 但是在实现时, 这些仲裁器所处的位置与传统算法不同, 输入端仲裁器位于各个 IPC 内, 所有输出端仲裁器位于一个集中的调度器中. *i*RGRR 算法的具体执行步骤如下:

(1)在每次执行过程开始前, 将所有输入和输出端都标志为未匹配.

(2)在每次迭代中:

Step1: 请求　每个输入端仲裁器有一个请求指针 r_i, r_i 指向当前优先选择的 VOQ. 如果一个未匹配的输入端有非空的 VOQ 等待发送信元, 从 r_i 指向的位置开始, 根据 round-robin 规则, 仲裁器选择某个 VOQ 的请求, 并且该 VOQ 对应的输出端空闲. 然后将此请求发送给相应的输出端仲裁器, r_i 当且仅当此请求在第一次迭代的 Step2 中被许可才更新, 等于当前被选择的 VOQ 端口号加 1(mod N), 否则不更新.

Step2: 许可　每个输出端仲裁器有一个许可指针 g_i, g_i 指向当前优先许可的输入端. 如果一个输出端仲裁器接收到一个或多个请求, 从 g_i 指向的位置开始, 根据 round-robin 规则, 许可某个输入端请求, 并发出一个许可信号. 调度器将所有这些信号经过简单的 OR 操作, 通知每个发出请求的输入端是否得到许可, 最后更新 g_i, 等于当前被许可的输入端口号加 1(mod N), 如果没有请求, g_i 不更新.

(3)在每个时隙结束时, 匹配结果配置 crossbar, 建立输入/输出端连接, 传送相应的信元.

*i*RGRR 每次迭代中也有两次信息交互, 但是总共只需要($\log N + 1$)比特, 即使对于 128 个端口, 请求信息只有 7 比特, 许可信息只有 1 比特, 这些少量的控制信息传送不会成为调度算法扩展中的瓶颈. *i*RGRR 算法不仅解决了传统调度算法可扩展性差的问题, 而且简化了调度器的设计, 提高了执行速度, 能够支持更多的端口数(64~128 甚至更多), 可以构建可扩展性强、高速、大容量的交换网络.

为了提高匹配效率,避免输入端向已经匹配的输出端发送请求,在 *i*RGRR 算法的“请求”步骤中,输入端仲裁器选择的某个非空 VOQ 的请求时要保证其对应的输出端尚未匹配.然而,输入端仲裁器无法得知输出端是否已经匹配.一种直观的方法是让调度器通知各个输入端,显然,这会将控制信息量增加到 $O(N)$ 数量级,仍然无法解决可扩展性问题.

为了更好地解决这个问题,我们使用一种简单的“输出端忙闲通知机制”,这只稍微增加了输入端仲裁器实现的空间复杂性,并不需要输入端与调度器之间交互其他额外控制信息.输出端忙和闲分别表示输出端已匹配和尚未匹配.这种机制基于如下事实:在一个时隙内,如果在某次迭代中一个输入端仲裁器向某个空闲的输出端发出请求,则该输出端此次迭代后一定忙,直到时隙结束.

在该机制中,每个输入端仲裁器与 N 条状态线相连,每条状态线只有两种状态:HIGH 和 LOW,分别表示一个输出端的忙和闲.在每个时隙开始和结束时刻,所有状态线都设置为 LOW.如果一个输入端仲裁器向某个输出端 j 发送了一个请求,发送该请求的同时将第 j 条状态线设置为 HIGH.如果状态线被设置为 HIGH,那么将一直保持到该时隙结束.这样,根据状态线指示的状态,输入端仲裁器可以屏蔽掉那些发送到“忙”的输出端的请求,从而避免了向已经匹配的输出端发送无用的请求.状态线只有两种简单的状态,并不用于传输数据,可以在每次迭代结束时同步设置这些状态线的状态.在具体实现时,由于输入/输出端口位于一个交换背板上,因此这些状态线可以方便地在背板上布线.

3.1 基于 *i*RGRR 算法的 IQ crossbar 结构

图 1 是基于 *i*RGRR 算法的 IQ crossbar 结构.输入端仲裁器位于每个 IPC 内,所有输出端仲裁器位于调度器内.假设所有的输入/输出端速率相同,所有 IPC 结构完全相同.信元到达过程可用离散时间随机过程表示,输入端 i 信元平均到达速率定义为业务流负载,用 λ_i 表示,且 $\lambda_i \leqslant 1$.图 2(a)展示了 *i*RGRR 算法在一个 4×4 IQ crossbar 中一次迭代的工作过程,图 2(b)列出了状态线的状态变化情况.

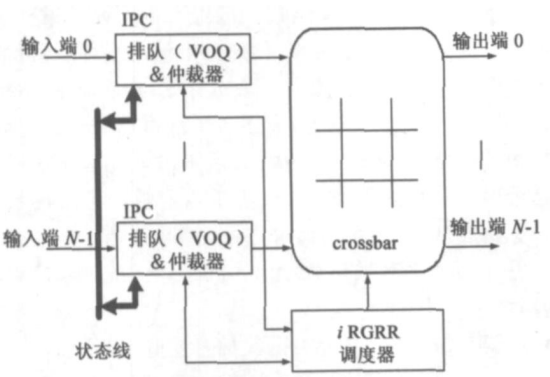
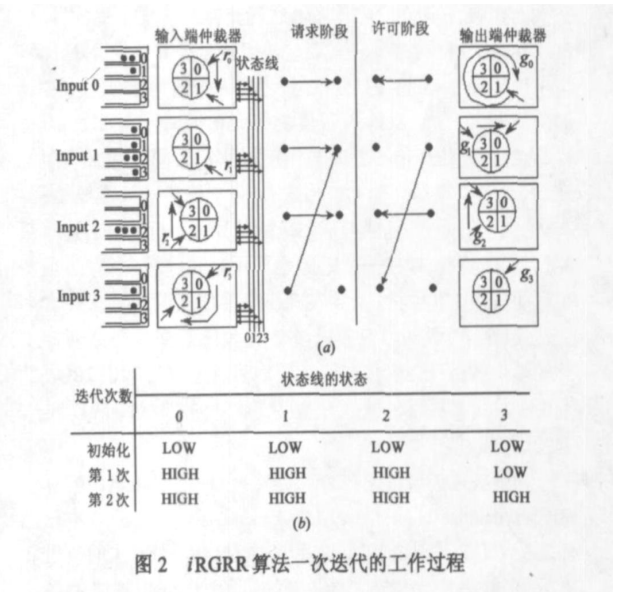


图 1 基于 *i*RGRR 算法的 IQ crossbar 结构

从图 2 中我们看到,在第 1 次迭代中,输入端 1、3 同时向输出端 1 发出请求,但只有输入端 3 得到许可,因此 r_3 才更新,r_1 不变.在开始时,状态线都是 LOW,第 1 次迭代后输入端仲裁器 0、2 将状态线 0、2 设置为 HIGH,输入端仲裁器 1 和 3 同时将状态线 1 设置为 HIGH.在第 2 次迭代(图示省略)中,虽然在输入端 1 中 $VOQ_{1,1}$ 和 $VOQ_{1,2}$ 非空,并且 $r_1 = 1$,但是由于状态线 1、2 为 HIGH 状态,它只会向输出端 3 发出请求,同时将状态线 3 设置为 HIGH,最终该请求被许可.在这个例子中,如果没有输出端忙闲通知机制,无论如何增加迭代次数,也不会增加新的匹配边.



迭代次数	状态线的状态			
	0	1	2	3
初始化	LOW	LOW	LOW	LOW
第 1 次	HIGH	HIGH	HIGH	LOW
第 2 次	HIGH	HIGH	HIGH	HIGH

(b)

图 2 *i*RGRR 算法一次迭代的工作过程

4 单个机架内 *i*RGRR 算法性能分析

对于 IQ crossbar 调度算法的性能分析,由于解析分析的困难性,计算机仿真是一种有效而广泛采用的研究手段[11].本节我们使用计算机仿真的方法对 *i*RGRR 算法进行吞吐量、时延等性能的全面分析.吞吐量是指在一个时隙内平均发送的信元数(×100%),也等于输出端口平均利用率.时延是指信元在输入队列中平均等待时间,单位为时隙.仿真工具采用文献[12]提出的高速交换网络仿真系统,仿真长度均为 100,000 个时隙.

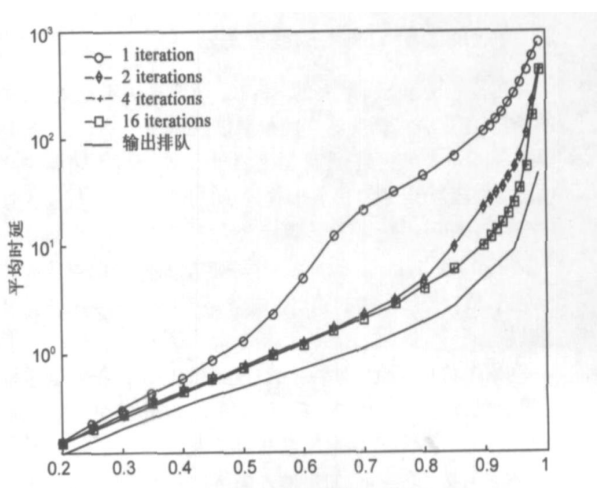


图 3 *i*RGRR 算法在不同迭代次数下的平均时延

4.1 迭代次数

*i*RGRR 算法采用多次迭代的方法尽量增加每次匹配的边数, 首先我们关心的是当采用多少次迭代时算法能够收敛. 在一个 16× 16 的 IQ crossbar 中, 图 3 展示了当信元按均匀独立同分布(i. i. d.)贝努利过程到达时, *i*RGRR 算法在不同迭代次数下的平均时延. 由于在同样的参数设置下, OQ crossbar 具有最优的性能, 因此, 我们将 OQ crossbar 的性能作为参考对象.

从图 3 中可以看出, 2 次迭代就能够大大改善时延性能, 4 次迭代几乎与 16 次迭代获得相同的平均时延. 这与 PIM、*i*SLIP 等算法具有相同的结果, 对于一个 $N\times N$ 的 IQ crossbar, $\log N$ 次迭代足以使 *i*RGRR 算法收敛.

4.2 吞吐量

在分析吞吐量和平均时延性能时, 我们将 *i*SLIP 算法作为比较的对象, *i*SLIP 算法在同类算法中被公认具有优良的性能, 并且取得了成功的应用. 在下面的仿真研究中, 如不明确指明, 两种算法一次执行过程包含 $\log N$ 次迭代.

在给出仿真结果之前, 我们可以直观地感觉到, 在相同条件下, *i*RGRR 算法难以超过 *i*SLIP 算法的吞吐量和时延性能. 因为 *i*SLIP 算法在每次迭代中, 每个输入端向调度器发出所有的请求, 只要其中一个被许可, 输入端就可以被匹配, 而 *i*RGRR 算法每次只发出一个请求, 被匹配的概率会降低. 在下面的仿真研究中, 我们特别关注两者之间的差距.

(1) 均匀业务流　文献[1]和[7]分别证明了 *i*SLIP 和 DRRM 算法在均匀 i. i. d. 业务流下都能获得 100% 的吞吐量. 由于 *i*RGRR 算法一次迭代的执行结果与 DRRM 完全相同, 根据图 3 所示, *i*RGRR 算法采用多次迭代还能改善时延性能, 因此, 在均匀 i. i. d. 业务流下, *i*RGRR 和 *i*SLIP 算法都能获得 100% 的吞吐量.

(2) 非均匀业务流　为了研究 *i*RGRR 算法在更复杂业务流到达情况下的吞吐量, 我们考虑如下通常采用的非均匀业务流模型.

(*a*) 业务流模型 1　所有输入端负载相同, 即 $\lambda_i=\lambda$, 按下式分布于各个 *VOQ*:

$$\lambda_{ij}=\begin{cases}\lambda\cdot\delta, & j=i\\ \lambda\cdot(1-\delta), & j=(i+1)\bmod N\end{cases},\ 0\leqslant i,j\leqslant N-1 \quad (1)$$

其中 λ_{ij} 表示从输入端 i 到输出端 j 的负载, $0\leqslant\delta\leqslant 1$ 为非均衡(unbalanced)因子[10].

表 1　*i*RGRR 和 *i*SLIP 算法在非均匀业务流模型 1 下的吞吐量

算法	非均衡因子				
	0.1	0.2	0.3	0.4	0.5
*i*SLIP	0.917	0.861	0.831	0.844	0.870
*i*RGRR	0.917	0.861	0.831	0.844	0.870

表 1 列出了一个 16× 16 规模 IQ crossbar 中, *i*RGRR 和 *i*SLIP 算法随 δ 变化时的吞吐量值, 两者相同.

(*b*) 业务流模型 2　所有输入端负载相同, 即 $\lambda_i=\lambda$, 按下式分布于各个 *VOQ*:

$$\lambda_{ij}=\begin{cases}\lambda\cdot\left(w+\dfrac{1+w}{N}\right), & j=i\\ \lambda\cdot\dfrac{1-w}{N}, & j\neq i\end{cases},\ 0\leqslant i,j\leqslant N-1 \quad (2)$$

其中 $0\leqslant w\leqslant 1$ 被称为非均匀(nonuniform)因子[9], 当 $w=0$ 时, 该业务流就是均匀业务流.

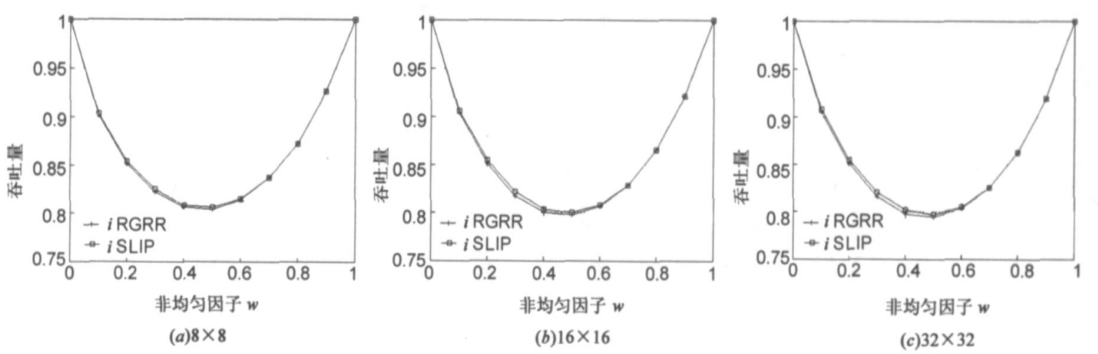


图 4　在不同规模大小的 IQ crossbar 中 *i*RGRR 和 *i*SLIP 算法的吞吐量

让 w 的值在 0 到 1 之间变化并记录下算法的吞吐量. 图 4 展示了在 8× 8、16× 16 和 32× 32 的 IQ crossbar 中 *i*RGRR 和 *i*SLIP 算法的吞吐量. 从图中可以看到, 在相同条件下, *i*RGRR 算法的吞吐量与 *i*SLIP 算法的十分接近, 最大差距不超过 0.5%, 同时 crossbar 规模的大小对算法的吞吐量影响不大.

在以上非均匀业务流模型下, *i*RGRR 算法拥有与 *i*SLIP 算法近似的吞吐量性能, 说明 *i*SLIP 算法执行过程中需要交互的控制信息中大量是冗余的, 不会对算法性能的提高有所帮助, 而 *i*RGRR 算法使用较少控制信息就可以获得良好的性能.

4.3 时延

图 5 展示了当信元按均匀 i. i. d. 贝努利过程到达时, *i*RGRR 和 *i*SLIP 算法在不同迭代次数下的平均时延. 图 6 展示了在 8× 8、16× 16 和 32× 32 的 IQ crossbar 中, *i*RGRR 和 *i*SLIP 算法在突发业务流到达下的平均时延, 突发流平均突发长度分别为 8、16 和 32(信元). 在相同条件下, 两者几乎没有任何差距. 从图 6 中还可以看出, *i*RGRR 算法的平均时延随着突发长度的增加成正比线性增加, 并且基本上不受 crossbar 规模的大小影响, 这与 *i*SLIP 算法的行为相似[1]. 以上结果表明, *i*RGRR 算法拥有与 *i*SLIP 算法近似的时延性能, 说明了 *i*RGRR 算法使用较少控制信息就可以获得良好的时延性能.

4.4 公平性

一个信元从到达 VOQ 队头时刻开始, 一直到被调度时

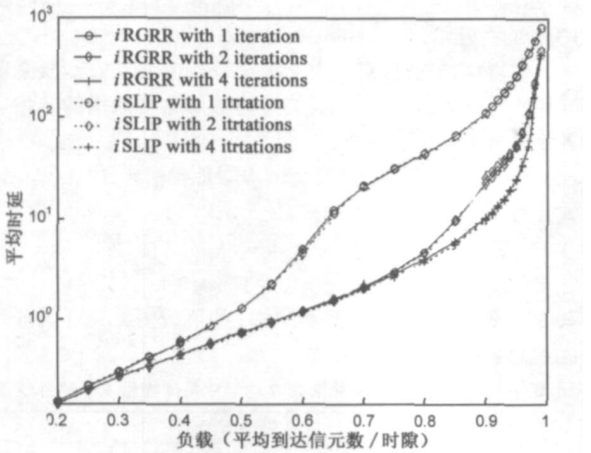


图 5 *i*RGRR 和 *i*SLIP 算法在均匀 i.i.d. 贝努利业务流下的平均时延

刻, 其间的时间间隔被称为调度时延(单位: 时隙) . crossbar 调度算法的公平性是指调度时延必须存在上限, 即不能产生“饿死” 现象, 否则算法就是不公平的. 显然, 调度时延越小说明算法的公平性越好.

在 *i*RGRR 算法中, 假设某个信元到达, 那么它的请求被输入端仲裁器选择发出至多需要等待指针更新 N 次, 而该输入端的每一个请求被输出端许可至多需要等待 N 个时隙, 即输入端仲裁器指针更新一次至多需要 N 个时隙. 所以, 该信元的请求被发出并得到许可至多需要等待 N^2 个时隙. 这个结论与 FIRM[5] 算法相同, 而 *i*SLIP 算法的调度时延上限为($N-1)^2+N^2$ 个时隙[1] . 因此, *i*RGRR 比 *i*SLIP 算法具有的更好的公平性.

5 多个机架内 *i*RGRR 算法性能分析

从扩展角度看, 由于受到体积和功耗的限制, 数十或上百个端口不可能放在同一个机架内, 可以放置于不同的机架

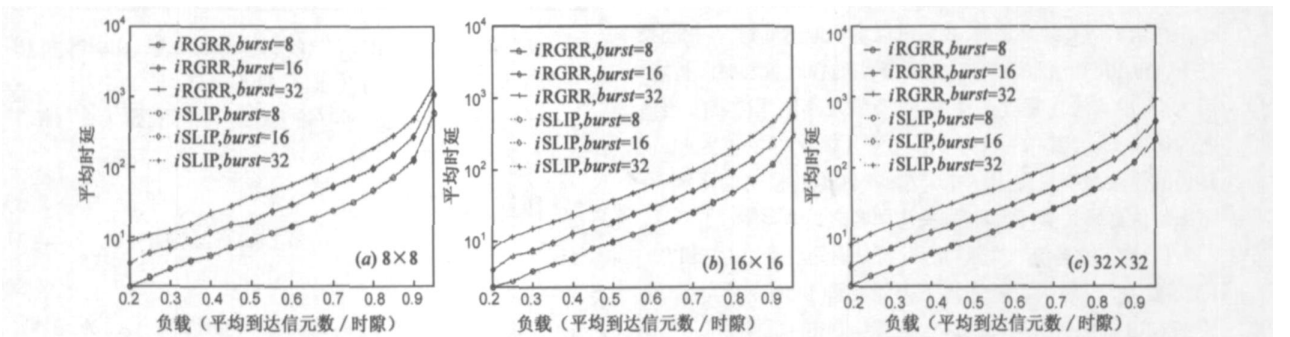


图 6 在不同规模大小的 IQ crossbar 中 *i*RGRR 和 *i*SLIP 算法在突发流下的平均时延

内[9, 13] . 然而这会对 *i*RGRR 算法中状态线的设置带来困难, 一种可行的方法是各个机架维护自身的状态线. 这样, 一个机架内的输入端仲裁器就无法获得其他机架内状态线上标志的状态. 例如, 考虑一个支持 16 个端口的 IQ 交换网络分布在两个机架上, 假设输入端 0~ 7 和输出端 0~ 7 位于机架 A, 其他的输入/输出端位于机架 B. 机架 A 和 B 各自维护 8 条状态线, 分别标志 A 和 B 内的输出端的忙闲状态. 这样, 在机架 A 内的输入端仲裁器会向 B 中某个“忙” 的输出端发出请求, 同样, 在机架 B 内也存在相同的情况, 输入端仲裁器也会向 A 中某个“忙” 的输出端发出请求. 很显然, 这样会降低 *i*RGRR 算法的性能. 这里只考虑一个 16 个端口的交换网络分布在 2 个和 4 个机架内的情况.

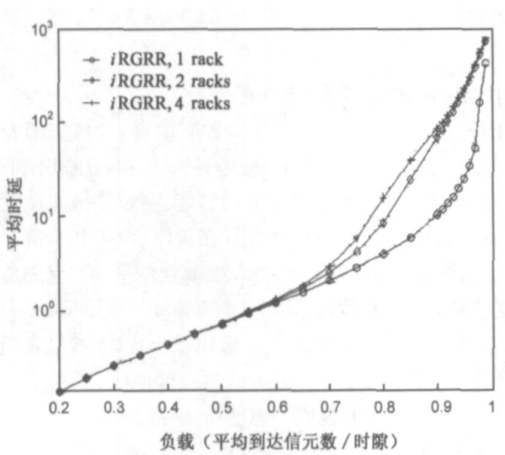


图 7 分布于 1、2 和 4 个机架内, *i*RGRR 算法在均匀 i.i.d. 贝努利业务流下的平均时延

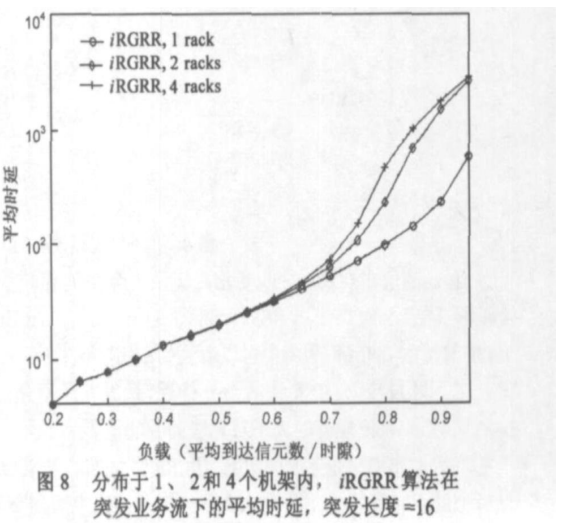


图 8 分布于 1、2 和 4 个机架内, *i*RGRR 算法在突发业务流下的平均时延, 突发长度 =16

在不同数目机架内, 图 7 比较了 *i*RGRR 算法在均匀 i. i. d. 贝努利业务流下的平均时延, 图 8 比较了 *i*RGRR 算法在突发业务流到达下的平均时延, 平均突发长度为 16. 当负载较大时($\lambda>0.6$), *i*RGRR 算法的平均时延随着机架数的增加迅

速增长,说明这个时候输入端发出的请求许多被浪费,性能下降.另一方面也说明了"输出端忙闲通知机制"对于改善 *iRGRR* 算法的性能非常有效.

6 结束语

本文提出了一种可扩展性强、高速、简单有效的 IQ crossbar 调度算法 *i*RGRR. 当前的调度算法普遍存在难以实现或可扩展性差的问题,问题的关键在于输入端与调度器交互的控制信息量过大,这些信息的传输时间占去调度算法执行时间的大部分,最终限制了算法的可扩展性. 与同类算法相比, *i*RGRR 算法最大的优势是将算法执行中需要的控制信息量复杂度从 $O(N)$ 级降到 $O(\log N)$, 克服了同类算法可扩展性差的缺陷, 能够支持 64 甚至 128 个 10Gbps 的高速端口. 仿真结果表明,在相同条件下, *i*RGRR 算法具有与 *i*SLIP 算法近似的吞吐量、时延性能,并且提供更好的公平性和更加易于硬件实现. 文献[11]给出了基于 *i*RGRR 算法的调度器的详细硬件实现和扩展方案,限于篇幅,这里不再赘述. 总之, *i*RGRR 算法具有良好的可扩展性,按照现有的 ASIC 技术,基于 *i*RGRR 算法的调度器能够支持具有太比特交换容量的交换机/路由器.

参考文献:

[1] N Mckeown. The iSLIP scheduling algorithm for input-queued switches[J]. IEEE/ACM Trans on Networking, 1999, 7(2): 188-200.

[2] T E Anderson, S S Owicki, J B Saxe, C P Thacker. High speed switch scheduling for local area networks[J]. ACM Transactions on Computer Systems, 1993, 11(4): 319-352.

[3] M J Karol, M Hluchyj, S Morgan. Input versus output queueing on a space-division packet switch[J]. IEEE Trans on Comm, 1987, 35: 1347-1356.

[4] N Mckeown, V Anantharam, J Walrand. Achieving 100% throughput in an input-queued switch[A]. Proceedings of IEEE Infocom[C]. San Francisco, USA, 1996, 1: 296-302.

[5] D Serpanos, P Antoniadis. FIRM: a class of distributed scheduling algorithms for high-speed ATM switches with multiple input qQueues[A]. Proceedings of IEEE Infocom[C]. Tel Aviv, Israel, 2000. 548-555.

[6] H J Chao, J S Park. Centralized contention resolution schemes for a large-capacity optical ATM switch[A]. Proceedings of IEEE ATM Workshop, Fairfax[C]. VA, USA, 1998. 11-16.

[7] Y Li, S Panwar, H J Chao. On the performance of a dual round-robin switch[A]. Proceedings of IEEE Infocom[C]. AK, USA, 2001. 1688-1697.

[8] E S Shin, V Mooney, G F Riley. Round-robin arbiter design and generation[A]. Proceedings of the International Symposium on System Synthesis (ISSS' 02)[C]. Kyoto, Japan, 2002. 243-248.

[9] C Minkenberg. Performance of iSLIP scheduling with large round-trip latency[A]. Proceedings of the IEEE 2003 Workshop on High Performance Switching and Routing[C]. Torino, Italy, 2003. 49-54.

[10] Marco Ajmone Marsan, Andrea Bianco, Enrica Filippi, Paolo Giaccone, Emilio Leonardi, Fabio Neri. On the behavior of input queuing switch architectures[J]. European Transactions on Telecommunications (ETT), 1999, 10(2): 111-124.

[11] 彭来献. 高速路由器交换体系与调度算法的研究[D]. 南京: 解放军理工大学, 2004, 5.

[12] 彭来献, 田畅, 郑少仁. 高速交换网络的建模与仿真[J]. 系统仿真学报, 2003, 15(10): 1474-1476.

[13] 彭来献, 李万林, 田畅, 郑少仁. 太比特路由器关键技术分析[J]. 电信科学, 2002, 18(3): 11-15.

作者简介:

彭来献　男, 1978 年 3 月出生于安徽阜阳, 博士, 讲师. 1999 年于解放军通信工程学院获工学学士, 2004 年 6 月在该院获通信与信息系统专业博士学位. 主要研究方向为高速路由器体系结构和高速交换网络. E-mail: penglaixian@sina.com

田　畅　男, 1963 年 2 月出生于山东青岛, 博士, 副教授, 中国电子学会高级会员. 主要从事宽带交换技术、网络安全和无线分组网的研究.