文章编号:1673-0291(2010)05-0008-06

面向并行安全网关流水线模型的无锁队列算法

高志民,姚 崎
(北京交通大学 计算机与信息技术学院,北京 100044)

摘 要:提出一种适用于并行安全网关流水线模型中共享数据缓冲区操作的无锁队列算法.与其他类似算法比较,该算法采用链表结构组织队列数据,避免了采用循环数组结构引起的缓冲区长度限制和内存浪费的问题;与通用的链表队列无锁算法比较,算法实现更为简洁,执行效率更高.证明了算法具有线性化和非阻塞特性.通过模拟试验,验证了算法在理想环境和各种实际应用环境中都具有较好的性能指标.

关键词:安全网关;流水线模型;生产者/消费者队列;无锁算法
中图分类号:TP309.1 **文献标志码**:A

A Lock-Free Queue Algorithm for Pipeline Model of Parallelism Security Gateway

GAO Zhimin, YAO Qi
(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: A lock-free queue algorithm for shared data buffer operation in pipeline model of parallelism security gateway is proposed in this paper. Comparing other similar algorithm that employs static cycle data array, dynamic linked list data structure is adopted in the proposed algorithm and thus restriction of data buffer length and memory waste is eliminated. Comparing the general lock-free algorithm for linked list queue, implementation of the proposed algorithm is simple and fast. Also, the linearizability and non-block attribution of the algorithm is proved in the paper. Finally, the results of simulation experiment show that performance of the new proposed algorithm is good not only in ideal experiment environment but also in all kinds of realistic experiment environment.

Key words: security gateway; pipe-line model; produce/consume queue; lock-free algorithm

随着网络速度的不断提升和网络攻击手段的日益复杂,对安全网关的处理能力要求也迅速增加.10 Gbps网络逐步向桌面延伸,40 Gbps骨干网已经进入到实用阶段,100 Gbps骨干网技术也正在成熟.同时,传统基于报文头部过滤的单一功能安全网关已经无法满足复杂的安全防护需求.安全网关逐步向着集成多种安全防护引擎、执行面向内容的深度过滤方向发展.为了适应这样的变化趋势,安全网关在设计和实现上必须具备更快的转发速率和更高计算性能.

基于多核处理器和多处理器系统的并行处理架构成为下一代高性能安全网关的必然选择.研究表明,在各种并行处理模型中,流水线模型在处理复杂的基于内容检测的安全网关中是一种高效的模型[1-3].一个或者多个安全引擎绑定运行在一个处理器上,执行特定的检测任务,这不仅可以提高处理器的CACHE局部性,而且大大减少了系统的调度开销.

收稿日期:2010-01-26;**修回日期**:2010-09-05
基金项目:国家"863"计划项目资助(2007AA01Z410);电子信息产业发展基金项目资助(2007-329)
作者简介:高志民(1973—),男,河南郑州人,博士生.**email**:zhimingao@163.com.

在流水线处理模型中,各个处理阶段间需通过共享的数据区来传递数据.共享数据区的读写效率将在很大程度上影响整个流水线系统的吞吐率.文献[4]研究指出,系统要达到 1 Gbps 的吞吐率,每秒钟需处理 1 488 095 个报文,每个报文的处理时间不能大于 672 ns,而传统的基于互斥锁的流水线队列操作算法执行一次读操作和一次写操作的时间约为 200 ns,这严重制约的系统整体处理能力的提升.

本文作者采用生产者-消费者队列模型来描述流水线各处理阶段之间的共享数据区读写问题,并在对比分析多种生产者-消费者队列算法的基础上,提出 FastList 无锁队列操作算法.该算法与其他类似算法相比,具有实现简单、效率高,且支持动态链表结构等优点.通过模拟实验证明 FastList 算法满足并发数据对象操作的线性化要求和非阻塞属性.

1　模型描述及评价指标

1.1　并行安全网关流水线模型

安全网关对每个数据报文的处理过程可以分为网络协议处理和安全检测处理两个阶段.网络协议处理阶段主要完成对报文的接收、校验和链路层协议处理(如:网桥协议处理)、网络层协议处理(如:路由协议处理)和基于报文头部的报文过滤(如:基于五元组的防火墙过滤)等.安全内容检测阶段主要完成对报文数据内容的各种安全性检查,如:攻击特征检查、病毒检查和恶意代码检查等.因此,在并行安全网关的流水线模型中,总体上采用两段式流水结构:网络处理引擎(Network Engine)NE 和安全处理引擎(Security Engine)SE,分别完成报文的网络协议处理和安全检测处理.但是,在实际过程中,NE 处理主要是相对简单的查表操作,处理时间短;而 SE 处理过程则包括数据流重组、字符串匹配、正则表达式匹配等耗时的操作,处理时间较长.因此,根据不均衡流水线模型提高吞吐率的解决策略,在并行安全网关中,一般会并发执行多个 SE 引擎,最终的处理模如图 1 所示.

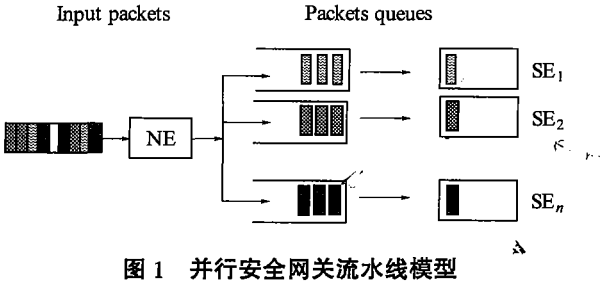


图 1　并行安全网关流水线模型

Fig.1　Pipe-line model of parallelism security gateway

上述模型中,NE 除了完成网络协议处理外,还有一个重要工作是将输入报文进行分流并提交给不同的 SE 进行下一步安全检测处理.由于多数 SE 处理过程都是要基于流的,因此分流算法采用基于五元组的 HASH 函数

$$h:\ U \rightarrow \{1,2,\cdots,n\},$$

其中:U 为数据报文的五元组 key 值全域;n 为网关中 SE 引擎的个数.分流后的数据报文被放置在每个 SE 独立维护的报文队列中等待处理.

1.2　生产者/消费者队列

在图 1 所示的流水线模型中,报文队列是两个流水段之间的共享数据结构.这是一个典型的生产者/消费者(producer/consumer)队列结构,采用下述模型进行描述.定义 Ω 代表整个生产者/消费者系统,Ω 由 5 个元素构成:

$$\Omega:\{P,\ C,\ Q,\ \text{consume},\ \text{produce}\},$$

其中:P 为生产者集合,$|P|$为生产者个数,$P=\{p_i \mid i \in [1,\ |P|]\}$,集合中的每个元素 p_i 代表一个独立运行的生产者线程,整个系统中要求至少包含一个生产者线程,即:$|P| \geqslant 1$;C 为消费者集合,$|C|$为消费者个数,$C=\{c_i \mid i \in [1,\ |C|]\}$,集合中的每个元素 c_i 代表一个独立运行的消费者线程,整个系统中要求至少包含一个消费者线程,即:$|C| \geqslant 1$;Q 为共享数据队列集合,$|Q|$为队列个数,$Q=\{q_i \mid i \in [1,\ |Q|]\}$,集合中每个元素 q_i 表示为一个独立的队列对象;produce 为生产操作,生产者线程 p_i 向数据队列对象 q_j 中放置新的数据元素 x 的操作记为:$\langle q_j.\text{produce}(x),\ p_i\rangle$;consume 为消费操作,消费者线程 c_i 从数据队列对象 q_j 中获取数据元素 y 所执行的操作记为:$\langle q_j.\text{consume}(y),\ c_i\rangle$.

对于某个给定的流水线系统,上述生产者/消费者模型中的 P、C、Q 是确定的,需求解的问题为:通过优化设计 produce 和 consume 两个操作算法使其满足下面的评价指标.

1.3　队列操作算法评价指标

对算法的评价指标有很多,根据在并行安全网关流水线模型中的实际应用要求,我们为生产者/消费者队列操作算法定义了 3 个评价指标.

指标 1:算法在并行执行环境中的正确性(correctness property).

从非形式化的角度解释,正确性指当并行执行针对共享数据对象的操作算法时,执行过程是安全的,总是能够得到预定义的执行结果;被操作的数据对象在任何时刻都是完整的,符合其数据结构的定义属性.

在形式化描述方面,对串行执行的程序正确性

的形式化定义和证明方法，有相对成熟的方法，比如：公理体系、指称体系等[5]. 而对于数据对象并行操作算法的正确性定义和证明仍然是研究热点. 本文中我们选择了目前在并行数据处理模型正确性描述和证明中最为严格同时被广泛使用的线性化理论(linearizability)[6-7]来刻画操作算法在并行执行环境中的正确性.

定义 1 如果算法的所有操作对数据对象状态的改变都是在该操作触发动作(invocation)和响应动作(response)之间的某个瞬间完成的，则称该算法实现在并行执行环境中是线性的.

对线性化的证明过程分为两个步骤：首先证明算法的任意顺序执行历史(sequential history)都满足被操作的数据对象的顺序化规约(sequential specification)；然后证明算法的每个操作实现都存在线性执行点(linearization points)，即从并行执行的其他操作者角度观察，该操作对数据对象状态的改变在线性执行点的瞬间完成.

指标 2：算法在并行执行环境中的活跃性(liveness property).

线性化描述了算法在并发执行环境下的正确性属性，但它没有反映并发执行的程序在访问共享数据对象时的阻塞问题. 例如：采用互斥锁和临界区的共享队列并发操作算法是线性化的[7]，但同一时刻只能允许一个线程获得锁或者进入临界区，其他试图访问共享数据的线程只能等待. 如果获得锁或者进入临界区的线程在执行过程中由于某种原因被阻塞了，那么所有其他等待线程同样被阻塞. 因此，我们引入另外一个度量指标——活跃性(liveness property). 从活跃性属性角度分类，并行算法可以分为两大类：阻塞算法(blocking algorithm)和非阻塞算法(non-blocking algorithm)，对非阻塞算法有可以根据算法特性分为不同的种类：Wait-free 算法、Lock-free 算法和 Obstruction-free 算法. 关于这些算法属性的详细定义，请参阅文献[8].

指标 3：算法的性能评价参数.

在算法的正确性与活跃性得到保障的条件下，算法并行执行效率是在设计和实现过程中另外一个重要的评价指标. 本文采用平均队列操作执行时间作为算法性能评价指标.

定义 2 平均队列操作执行时间 $\bar{\tau}$.

设算法中生产操作执行时间为 t_p，消费操作执行时间为 t_c，由于每个报文在流水线的处理过程中都会执行一次生产操作和一次消费操作，因此，我们定义一次队列操作执行时间为

$$\tau = t_p + t_c.$$

设生产者连续执行 n 次生产操作，消费者连续执行 n 次消费操作，则平均队列操作时间为

$$\bar{\tau} = \frac{\sum_{i=1}^{n}(t_{pi} + t_{ci})}{n}.$$

2 FastList 无锁队列算法

根据图 1 所示的并行网关流水线模型，整个报文队列数据缓冲区有 1 个生产者和 n 个消费者. 但是，由于缓冲区内每个报文队列之间是相互独立的，没有共享数据访问冲突问题. 每个报文队列自身仅存在 1 个生产者和 1 个消费者，因此，可以将问题简化为针对单生产者/单消费者(SP/SC)队列的操作算法设计.

2.1 相关算法分析

针对 SP/SC 队列的并发操作已经有很多经典算法. 最直接的算法的基于互斥锁(lock-based)的算法[4]，生产者和消费者作在操作队列之前获得锁，在操作完成之后释放锁. 这种算法的优点是简单、易实现，算法适应性比较广，但是锁操作的效率不高，而且算法是阻塞的，易产生死锁、活锁等问题. Lamport 在文献[9]中的研究指出，由于 SP/SC 系统的特殊性(只有一个生产者和一个消费者)，只需要对算法实现进行简单改进即可实现符合 Lock-free 属性的非阻塞操作算法(记为 Lamport 算法). Lamport 算法是 20 世纪 80 年代提出的，之后很长一段时间内，该算法被公认为解决 SP/SC 队列问题的最高效的算法. Giacomoni 等人在 PPoPP08 会议上发表了针对 Lamport 算法的优化算法——FastForward 算法[4]，通过对 Lamport 算法的改进，消除了生产操作中对队列 HEAD 指针的读写和消费操作中对队列 TAIL 指针的读写，使得算法在多处理器环境中执行时不会产生处理器 CACHE 频繁同步的问题，执行效率有了进一步的提升.

Lamport 算法和 FastForward 算法是关于 SP/SC 队列操作的经典算法，都满足 Lock-free 属性，可以在并行系统中高效执行. 但是，它们都采用静态循环数组结构来组织队列元素，这种结构的存储容量受限于数组的大小 L，当队列中的数据元素达到 L 个之后，生产操作就会被阻塞或者 新生产的数据会被丢弃. 对于遵循 burstness 规律的网络数据报文流来讲，确定合适的 L 是一件比较困难的事情. L 定义的太大会造成多数情况下存储资源的浪费，L 定义的过小，在突发流量的时间段内很容易产生丢包.

因此更适合安全网关流水线模型的队列结构应该是动态链表.

针对通用的多生产者/多消费者(MP/MC)动态链表队列的非阻塞算法研究成果非常多[10-12],其中以 Michael、Scott 等人提出的算法最为经典(以下简称 MS 算法),其已经被实现在 java 的 concurrent queue 对象的操作方法中.无论是 MS 算法还是其他非阻塞队列操作算法,都不是专门针对 SP/SC 队列而设计的,其面向的是更为通用的 MP/MC 情况.因此,在算法设计中采用了比较耗时的 CAS、LL/ST 等原子处理器指令,来保证链表指针修改的原子性;同时为了消除 ABA 问题,采用"胖指针(fat pointer)"结构;为了消除多个生产者之间和多个消费者之间的互斥问题,采用了多次比较判断操作.这些处理都增加了算法的复杂性和执行时间,而实际上这些问题在 SP/SC 队列中都是没有的.

2.2 FastList 算法

针对 SP/SC 队列的特点,借鉴 Lamport 和 FastForward 两个算法的设计思想,提出面向动态链表结构的无锁队列算法——FastList 算法见图 2.

```
structure data_t { packet: pointer to packet, next: pointer
to data_t}
Structure queue_t { head: pointer to data_t, tail: pointer to
data_t }
queue : global variable of queue_t type

produce_FastList (new_data: pointer of data_t )
{
P1:        new_data->next = NULL;
P2:        atom_set(queue.tail->next , new_data);
P3:        queue.tail = new_data;
}

consume_FastList (ret_data: pointer of data_t )
{
C1:        ret_data = NULL;
C2:        if (queue.head->next == NULL )
C3:              return;
C4:        ret_data = queue.head;
C5:        atomic_set(queue.head , queue.head->next);
C6:        ret_data->next = NULL;
C7:        return;
}
```

图 2　Fast list 算法实现

Fig.2　Implementation of fast list algorithm

在 FastList 算法中,借鉴了 FastForward 算法的思想,在报文队列中始终保留一个数据元素,在 produce 操作中只读写队列的 TAIL 指针,在 consume 操作中只读写队列的 HEAD 指针,从而消除了由于生产者线程和消费者线程访问共享数据变量引起的频繁 CACHE 同步开销,提升算法执行效率.为了保证算法正确执行,在报文队列初始化的时候,会为队列插入一个空的"哨兵数据元素",消费者获得"哨兵数据"后不对其进行任何后续处理,直接释放并获取下一个数据.

下面对 FastList 的线性化和非阻塞特性进行证明.

2.3 算法线性化证明

根据 1.3 节描述的线性化证明方法,首先证明 FastList 算法在顺序执行情况下,满足队列对象的顺序规约(sequence specification),即:

定理 1　在基于 FastList 算法的 SP/SC 队列的任意顺序化执行历史 H 中,如果下述操作的偏序关系成立

$$\langle q.\ \text{consume}(x), c_1\rangle <_H \langle q.\ \text{consume}(y), c_1\rangle,$$

则必然有

$$\langle q.\ \text{produce}(x), p_1\rangle <_H \langle q.\ \text{produce}(y), p_1\rangle.$$

然后,需要证明 FastList 算法实现的每个操作过程存在线性执行点(linearization point),即从系统中并发执行的其他操作角度观察,每个操作对数据对象状态的改变都是在该操作执行过程的某个瞬间完成的.

首先定义数据对象的状态变量 s.

采用队列 q 中由 next 指针顺序连接的数据元素集合作为 q 的状态描述变量.

$$s = \{x_1 * x_2 * \cdots * x_n * \text{null}\},$$

式中:$x_1, x_2, \cdots, x_n$ 表示队列中的数据元素,$*$ 表示数据元素之间由指针相连接,队列最后一个数据元素 x_n 的 next 指针必须指向 null,根据算法实现,队列中至少存在一个数据元素,因此 $n \geqslant 1$.

然后定义算法的每个操作过程对数据对象的状态转换.消费操作引起的数据对象状态转换有两种:当队列 q 中仅有一个数据元素时,操作失败返回 null,队列状态不变;当队列 q 中存在两个以上元素是,操作成功返回当前队列的第一个数据元素,并将该元素从队列中删除.

$q.\text{consume}(\text{ret})$:

$$s = \{x_1 * \text{null}\} \rightarrow \text{ret} = \text{null} \wedge s' = \{x_1 * \text{null}\} \quad (1)$$

$$s = \{x_1 * x_2 * \cdots * x_n * \text{null}\} \rightarrow \text{ret} = x_1 \wedge s' = \{x_2 * x_3 \cdots * x_n * \text{null}\} \quad (2)$$

生产操作引起的数据对象状态转换不依赖与队列的初始状态,在任何情况下其都将新的数据元素连接到队列最后:

$q.\text{produce}(x')$:

$$s = \{x_1 * x_2 * \cdots * x_n * \text{null}\} \rightarrow s' = \{x_1 * x_2 * \cdots * x_n * x' * \text{null}\} \quad (3)$$

定理2 在FastList算法的程序实现中，C2和C5是q.consume(ret)操作的线性执行点.其中C2瞬间完成式(1)的状态转换，C5瞬间完成式(2)的状态转换.

定理3 在FastList算法的程序实现中，P2是q.produce(x')操作的线性执行点，瞬间完成式(3)的状态转换.

由于篇幅关系，不对上述定理进行展开证明，详细的证明过程可以参考文献[13].

根据定理1～3可得，FastList算法针对SP/SC队列的操作是线性的.

2.4 算法非阻塞特性

在FastList算法的实现程序中，没有任何的循环结构和等待操作，在任何情况下，算法的所有执行实例都能够保证在有限的执行步骤内完成，因此FastList算法具有wait-free的非阻塞特性.

2.5 算法扩展性

FastList算法仅在SP/SC队列操作中符合线性化的要求，如果系统有存在多个生产者或者多个消费者，上述证明是不成立的.因为在SP/SC系统中只有一个生产者线程p_1和一个消费者线程c_1，所以并发执行的操作只存在于一个q. produce(x')操作实例和一个q. consume(ret)操作实例之间.在算法的实现中q. produce(x')只读写q. tail指针，q. consume(ret)只读写q. head指针，因此这两个指针数据的状态不对系统中并发执行的其他操作产生影响，可分别视为q. produce(x')操作和q. consume(ret)操作的内部状态，没有出现在状态描述变量s中.当系统中存在多个生产者或者多个消费者的时候，s中就必须包含q. head和q. tail指针的状态，这种情况下，在FastList算法的实现中是找不到线性化执行点的，算法并发执行的正确性也无法得到保证.

但是，在并行网关流水线模型中，每个独立的数据对象q_i实际上都只有一个生产者和一个消费者，因此可以认为整个报文队列数据区的数据对象是由多个线性的SP/SC队列对象组合实现的.根据线性化对象的组合特性，可知FastList算法是适用的.

3 模拟试验

3.1 试验环境

通过模拟试验获得算法的性能衡量指标.试验中选取双Intel Xeon X3210 4核处理器作为硬件平台，共有8个独立处理器核；采用Linux 2.6.24.4内核构建软件环境.为了减少操作系统调度等环境因素对试验的影响，获取算法的最佳性能指标，首先构建理想测试环境，对8个处理器核进行了功能划分，如表1.

表1 理想测试环境中处理器配置

Tab.1 Configure of CPU in ideal experiment environment

处理器核	执行功能	执行环境
CORE0	试验控制与结果分析	完整的Linux系统
CORE1	模拟生产者操作	独占处理器
CORE2-7	模拟消费者操作	独占处理器

在理想环境中，CORE0处理器作为试验控制单元，运行完整的Linux操作系统；CORE1-7作为算法执行单元，通过“处理器热插拔”技术从系统调度表中删除，处于无调度空闲状态，在该状态下处理器不参与系统调度，也不响应任何中断，仅循环执行空闲等待代码.通过修改代码，使CORE1循环执行生产者操作，CORE2-7循环执行消费者操作，从而构建独占处理器的理想测试环境.

在理想测试环境中测试得到的算法性能指标的最佳值.在实际的并行安全网关的应用中，完全独占处理器资源的情况非常少，更多的情况是生产者和消费者运行在完整操作系统环境的核心层或者应用层.为验证FastList算法的适用性，分别构建3种实际应用试验环境，如表2.

表2 实际应用测试环境配置

Tab.2 Configure of realistic experiment

试验环境	环境说明
核心层流水线	生产者、消费者均运行在操作系统核心层
应用层流水线	生产者、消费者均运行在操作系统应用层
交叉式流水线	生产者运行在操作系统核心层， 消费者运行在操作系统应用层

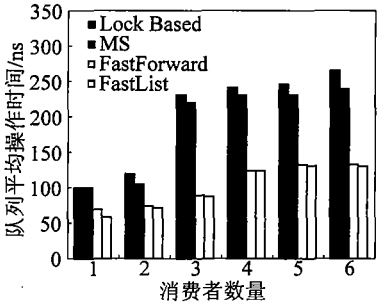
在这3种测试环境中，每个生产者和消费者仍然与某个处理器核进行绑定，处理器核的执行功能分配与理想试验环境中相同.

为了较为真实的模拟流水线各个阶段处理过程，在生产者线程和消费者线程的两次操作之间适当增加延迟.延迟时间与生产者和消费者数量成比例，最终使得报文队列长度处于较为均衡的状态.在计算每操作执行时间时，并不计算延迟时间.

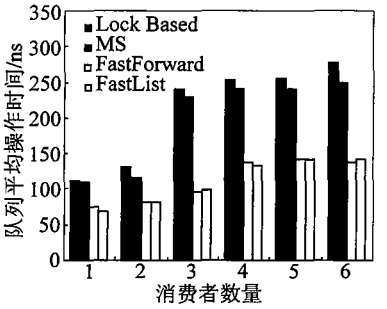
3.2 试验结果与分析

在上述4种测试环境下对Lock-based、MS、FastForward、FastList等4种队列操作算法进行了性能测试，测试中生产者和消费者各自连续执行100万次生产操作和消费操作，然后根据1.3节的定义计算队列平均操作时间.测试结果如图3.

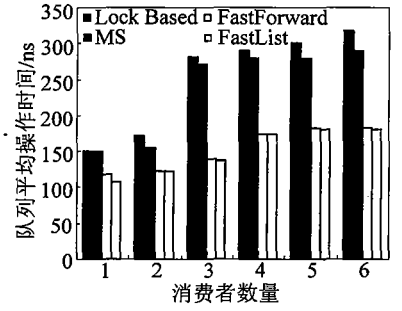
试验结果表明：基于互斥锁的lock-based算法的平均队列操作时间最长；MS算法虽然具有更广



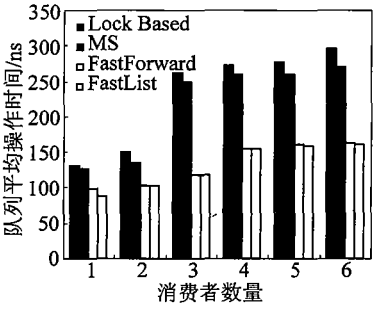
(a)理想测试环境



(b) 核心层流水线环境



(c) 应用层流水线环境



(d) 交叉流水线环境

图 3　各种环境下算法性能测试结果

Fig.3　Test results of algorithms in all experiment environments

泛的适用性,但是由于其处理过程复杂,采用的原子指令比较耗时,因此在流水线模型的队列操作中并不能获得较好的性能,其平均队列操作时间与 lock-based 算法基本相当;FastForward 算法和 FastList 算法在执行性能上明显优于其他两种算法.在 4 种测试环境中,理想测试环境下的各个算法性能指标最佳;核心层流水线环境下的测试结果与理想环境差别不大,这主要因为核心执行的线程优先级比较高,其受到操作系统调度的干扰较小;应用层流水线环境下各算法的处理性能最差,因为应用程序的运行更加容易被系统中断、调度等操作影响.在各种测试环境下,各个算法性能指标的对比趋势是基本相同的.

4　结论

针对并行安全网关的流水线处理模型,提出了一种无锁队列算法——FastList,该算法采用动态链表结构组织队列元素,克服了以往类似算法采用循环数组结构带来的存储空间限制和内存浪费问题;同时由于没有采用 CAS 等耗时较长的原子指令和"胖指针"结构,使其较通用链表队列无锁算法实现更为简洁,执行效率更高.通过证明实验,在并行计算环境中,FastList 算法满足 SP/SC 队列的线性化属性要求,且符合 wait-free 的无阻塞算法属性.通过模拟试验,验证了该算法在理想环境和各种实际应用环境中都具有较好的性能指标.经过扩展后,算法能够适用于各种流水线并行处理模型中,由多个 SP/SC 队列组合构成的共享数据缓冲区的维护.

参考文献:

[1] Guo Danhua, Liao Guangdeng, Bhuyan L N, et al. A Scalable Multithreaded L7-filter Design for Multi-Core Servers[C] // ANCS'08, San Jose. California, USA: ACM, 2008: 60 - 68.

[2] Schuff D L, Choe Y R, Pai V S. Conservative vs. Optimistic Parallelization of Stateful Network Intrusion Detection[C] // PPoPP'07, San Jose. California, USA: IEEE, 2007:138 - 139.

[3] Wu Qiang, Wolf Tilman. On Runtime Management in Multi-Core Packet Processing Systems[C] // ANCS'08, San Jose. California, USA: ACM, 2008: 69 - 78.

[4] Giacomoni J, Moseley T, Vachharajani M. FastForward for Efficient Pipeline Parallelism: A Cache-Optimized Concurrent Lock-Free Queue[C] // PPoPP'08, Salt Lake City, USA: ACM, 2008: 43 - 52.

[5] 古天龙. 软件开发的形式化方法[M]. 北京:高等教育出版社,2005.
GUO Tianlong. Formal Methods for Software Development[M]. Beijing: Higher Education Press, 2005. (in Chinese)

[6] Herlihy M, Wing J M. Linearizability: A Correctness Condition for Concurrent Objects[J]. ACM Transactions on Programming Languages and Systems, 1990,12(3):463 - 492.

(下转第 19 页)

货营销等领域,将有极大的应用价值和研究空间.下一步我们将进一步开展 MapReduce 对大规模视音频数据的处理研究.

参考文献:

[1] Michael Armbrust, Armando Fox. Above the Clouds: A Berkeley View of Cloud Computing[EO/BL].(2009)http://www.EECS.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[2] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337-1348.
CHEN Kang, ZHENG Weimin. Cloud Computing: System Instances and Current Research[J]. Journal of Software, 2009,20(5):1337-1348.(in Chinese)

[3] Jeffrey Dean, Sanjay Ghemawat. MapReduce: A Flexible Data Processing Tool[J]. Communications of the ACM, 2010,53(1): 72-77.

[4] Michael Stonebraker, Daniel Abadi, David J. DeWitt. MapReduce and Parallel DBMSs: Friends or Foes? [J]. Communications of the ACM, 2010, 53(1):64-71.

[5] Pavlo A, Paulson E, Rasin A. A Comparison of Approaches To Large-Scale Data Analysis[C]// Proceedings of the ACM SIGMOD International Conference. New York: ACM Press, 2009:165-178.

[6] Nanda S, Chiueh T. A Survey of Virtualization Technologies[R]. Tech. Rep. 179. New York: Stony Brook University, 2005.

[7] Kozuch M A, Ryan M P. Tashi: Location-Aware Cluster Management[C]// Proceedings of the ACM. Barcelona: ACDC, 2009:43-48.

[8] Tom White. Hadoop: Definition Guide[M].U.S.A., O'REILLY, 2009:1-299.

(上接第 13 页)

[7] Herlihy M, Shavit N. The Art of Multiprocessor Programming[M]. USA: Elsevier Inc, 2008.

[8] Paul E. McKenney. Exploiting Deferred Destruction: An Analysis of Read-Copy-Update Techniques in Operating System Kernels[D]. U.S.: School of Science & Engineering at Oregon Health & Science University, 2004: 79-83.

[9] Lamport L. Specifying Concurrent Program Modules[J]. ACM Transactions on Programming Languages and Systems, 1983, 5(2):190-222.

[10] Ladan Mozes E, Shavit N. An Optimistic Approach to Lock-Free FIFO Queues[C]// Proceedings of the 18th International Conference on Distributed Computing, Springer, 2004:117-131.

[11] Moir M, Nussbaum D, Shalev O, et al. Using Elimination to Implement Scalable and Lock-Free FIFO Queues [C]//SPAA'05: Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, New York, NY, USA: ACM Press, 2005: 253-262.

[12] Michaël M, Scott M L. Nonblocking Algorithms and Preemption-Safe Locking on Multiprogrammed Shared: Memory Multiprocessors[J]. Journal of Parallel and Distributed Computing, 1998, 51(1):1-26.

[13] 姚崎. 并行安全网关若干关键问题研究[D]. 北京: 北京交通大学,2010:87-95.
YAO Qi. Research on Some Key Techniques in Parallelism Security Gateway[D]. Beijing: Beijing Jiaotong University, 2010: 87-95. (in Chinese)