HIGH PERFORMANCE DEEP PACKET INSPECTION ON MULTI-CORE PLATFORM

Wang Cong, Joe Morris, Wang Xiaojun

Dublin City University, Dublin, Ireland
cong.wang3@dcu.ie, joseph.morris@computing.dcu.ie, xiaojun.wang@dcu.ie

Abstract

Deep Packet Inspection (DPI) provides the ability to perform Quality of Service (QoS) and Intrusion Detection on network packets. But since the explosive growth of Intrnet, performance and scalability issues have been raised due to the gap between network and end-system speeds. This article describles how a desirable DPI system with multi-gigabits throughput and good scalability should be like by exploiting parallelism on Network Interface Card, network stack and user applications. *Connection-based Parallelism, Affinity-based Scheduling* and *Lock-free Data Structure* are the main technologies introduced to alleviate the performance and scalability issues. A common DPI application *L7-Filter* is used as an example to illustrate the applicaiton level parallelism.

Keywords: DPI; Multi-Core; Parallelism; Lock-Free Data Structure

1 Introduction

Deep Packet Inspection (DPI) provides the ability to examine the header and payload of packets. It allows that specified actions such as guaranteeing the transmitting quality and intrusion detecting, can be performed on them. As a popular DPI application, the L7-Filter[1] is widely used in providing traffic controlling in assocation with Linux QoS architecture.

The explosive growth of Internet, in terms of both hosts and network speed, results in a rapidly increasing network load. Addtionally, network speed tends to increase faster than CPU and memory speed, consequently casuing the gap between network and end-system speed. For instance, 10Gbps Ethernet[7] have been widely used by industrial community, while general-purpose processors which can fulfill the requirement for driving a 10Gbps throughput are unlikely to be available anytime soon. Therefore, in order to maintain a high level of throughput and low latency, the conventional solution is to use multiple processors for packet processing simultaneously.

978-1-4244-4591-2/09/$25.00 ©2009 IEEE

While elevating the performance of applications, the introduction of parallelism brings the scalability issues. Research results[2] have shown that on web server running a SPECweb2005 workload, throughput scales only 4.8x on eight cores. The key to scalability is to ensure system and hardware are designed to fully expoilt parallelism at all levels, i.e. hardware, Operating System and application.

DPI application must deal with huge numbers of connections simultaneously. Thus, C*onnection-based Parallelism*[4] is believed to be the most effective approach to exploit parallelism at hardware level. While elevating the utilization of CPUs, connection-based parallelism on NICs may lead to a load imbalance between processes[3] due to the unpredictability of the load in connections. However, OS schedulers can compensate the load imbalance by migrating processes between cores, but unfortunately at the same time produce scheduling overhead and cache-miss[8]. This problem can be solved by binding processes to cores, the so called *Affinity-based Scheduling*. However, even with the help of the above technologies, it is still challenging to implement a user application with good performance and scalability, mainly due to the communication cost introduced by using locks. To attack this challenge, *Lock-free Data Structure*[10] needs to be incorporated into the implementation to eliminate the usage of locks.

The remainder of this article is organized as follows. Section 2, 3, 4 describe the necessity of applying *Connection-based Parallelism, Affinity-based Scheduling* and *Lock-free Data Structure* respectively, by showing comparsions between old-fashioned and newly proposed practices. Section 5 introduces the related work in parallelism enabled DPI application. Finally, conclusions and future work are provided in Section 6.

2 Connection-based Parallelism

Parallelism at hardware level can both boost the throughput of NICs and decrease the latency of network communication. Figure 1 shows the old-fashioned architecture of NICs. In this architecture, NIC uses Direct Memory Access (DMA) to write

the received packet into a buffer and interrupts the system. The interrupted CPU then executes the functions in the NIC driver to process the packets. As comparison, Figure 2 shows a *Connection-based Parallelism* architecture. In this architecture, a hash function is used to determine which CPU should be interrupted, based on the connection the received packet belongs to.

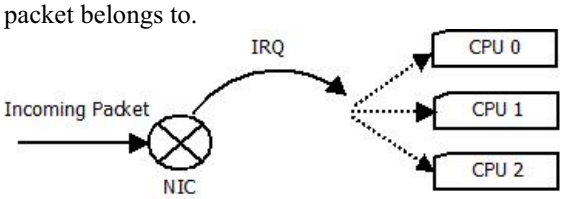

Figure 1. Old-fashioned NIC architecture

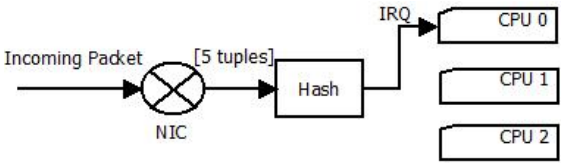

Figure 2. New architecture with *Connection-based Parallelism* enabled

Connection-based Parallelism is able to improve the network performance mainly due to:

1) It distributes packets to cores, thus elevate the ultilization and ensure no cores are idle while others are heavily loaded.
2) It allows the network protocol processing for packets which belong to the same connection can be executed on the same core, thus ensure cache-reloading and locks are less likely to happen.

The Connection-based Parallelism was first productized by Microsoft in their *Receive-Side Scaling (RSS)* technology[9], in which a particular core was designated to execute the processing of packets from the same connection. Comparing with RSS, the NAPI architecure in Linux Kernel only dispatch packets on a packet basis rather than on connection basis. Performance evaluation of *RSS* technology has been done using Web Bench[16] and the results are shown in Figure 3.

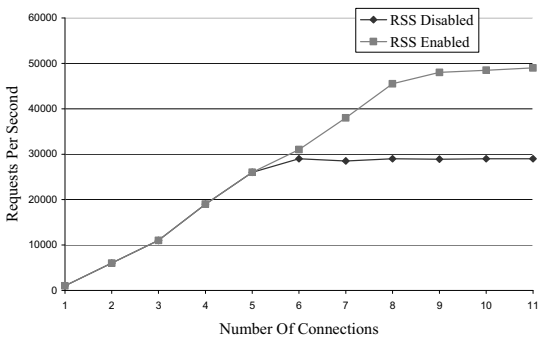

Figure 3. Benchmarks of RSS on a Symmetric Multiprocessing (SMP) platform

3 Affinity-based Scheduling

While using *RSS* or other *Connection-based Parallelism* technologies can boost the throughput and elevate the ultilization of cores, it may lead to a load imbalance due to the unpredictability of the load in connections. OS scheduler can compensate the imbalance by migrating threads from busy cores to idle cores. Unfortunately, the migration always comes with overhead of scheduling and cache-miss caused by moving data between cores. A trade-off between load imbalance and less scheduling overhead has to be made.

Taking a popular DPI application the L7-Filter as an example. Figure 4 shows the architecture of L7-Filter, which consists of a Scheduling Thread and a Matching Thread. By introducing load balancing mechanism in Scheduling Thread and binding each Matching Thread to a particular core, fundamental load balance can be guaranteed, thus migration from scheduler is no more needed. The key to this solution is *Affinity-based Scheduling* which can promise a thread will always be executed on the designated core. The new *Affinity-based Scheduling* enabled architecture of L7-Filter is shown in Figure 5. The cache performance of a multi-threaded L7-filter running on a kernel incorporating *Affinity-based Scheduling* has been evaluated by Guo[5] and is shown in Figure 6.

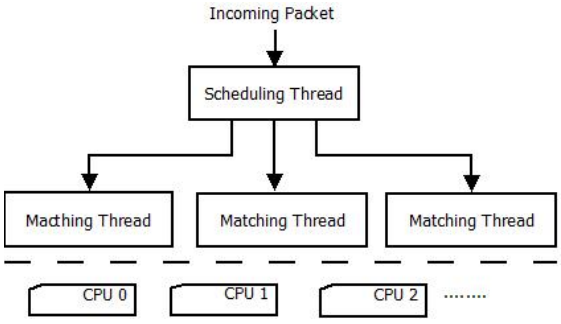

Figure 4. L7-Filter architecture

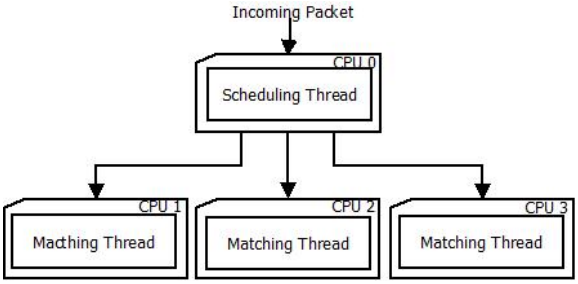

Figure 5. L7-Filter architecture with *Affinity-based Scheduling* enabled

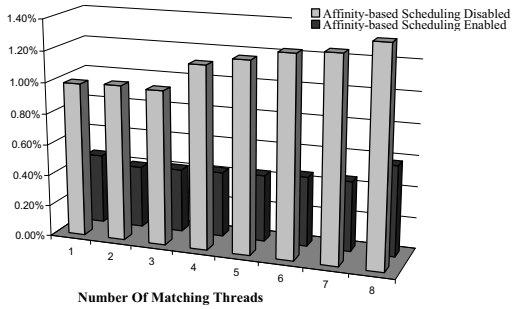

Figure 6. L2 cache miss rate of a multi-threaded L7-filter on a kernel using *Affinity-based Scheduling*

The old Linux scheduler has a particular weakness which causes random migration of threads between cores[6]. Since Linux 2.6, the old O(n) scheduler was replaced by a new O(1) scheduler which introduced the mechanism of CPU affinity by providing two new system calls:

```
long sched_setaffinity(pid_t pid, unsigned int len,
unsigned long *user_mask_ptr);
long sched_getaffinity(pid_t pid, unsigned int len,
unsigned long *user_mask_ptr);
```

4 Lock-free Data Structure

By introducing the above technologies, parallelism can be expoited at hardware level and operating system level. Addtionally, the DPI application itself has to fully exploit parallelism as well. In a typical multi-threaded DPI application, shared data structures are generally protected by locks, semaphores or other blocking synchronisers. *Lock-free Data Structures* are data structures that make no use of locks. Instead, they use *compare-and-set* (*CAS*) [10] instructions or classes based on *CAS* instructions. By introducing *Lock-free Data Structure*, the DPI application can achieve:

1) Eliminating locks, thus deadlock is no more a concern.
2) High concurrency while accessing different items.
3) Good scalability as number of processes increase.

5 Related Work

At the time of this writing, only a few parallel versions of DPI application are proposed. A common weakness among these proposals is the absence of inherent parallelism in all levels.

Intel[12] has made their attempt in parallelizing a popular DPI application SNORT IDS by reconstructing its architecture to incorporate *pipe-line* and *flow-pinning* technologies. Additionally, Some practices have been done in parallelling L7-Filter using *Affinity-based Scheduling* and *Lock-free Data Structure* by Wang, J.[11] and Guo, D.[5]. But these modifications only remain at application level. E. Lemoine[4] in his paper proposed a solution for hardware level parallelism by introducing packet classification in the interrupt dispatching mechanism of NICs. Again, this modification is limited in network interface cards, i.e. hardware level.

Despite of the above attempts in parallelizing DPI application, there are various other solutions have been proposed. Instead of exploiting parallelism, they choose other mechanisms, such as advanced algrithom for *Regular Expression*[13][15] and *Hardware Accelaration*[14], to improve the performance of DPI applications. In general, none of them can be proved to have good scalability with multi-cores.

6 Conclusions

The key to ensure a DPI platform has high performance and scales well with multi-cores is to ensure the layers of the platform, from hardware level to application level, are all designed to fully exploit parallelism. At the time of this writing, there are no well known platforms that fulfill this requirement. In future, a more advanced implementation of DPI application can be made which takes advantage of all the proposed technologies.

References

[1] Application Layer Packet Classifier for Linux (L7-filter), http:// l7-filter.sourceforge.net/.
[2] B. Veal, et al., “Performance Scalability of a Multi-core Web Server”, ANCS 2007.
[3] E. M. Nahum, D. J. Yates, J. F. Kurose, and D. Towsley. Performance issues in parallelized network protocols. In First USENIX Symposium on Operating Systems Design and Implementation (OSDI), Monterey, CA, November 1994.
[4] E. Lemoine, C. Pham, L. Lefèvre, "Packet Classification in the NIC for Improved SMP-based Internet Servers", Proceedings of the 3rd IEEE International Conference on Networking (ICN 2004).
[5] Guo, D., Liao, G., Bhuyan, L. N., Liu, B., and Ding, J. J. 2008. A scalable multithreaded L7-filter design for multi-core servers. In Proceedings of the 4th ACM/IEEE Symposium on Architectures For Networking.
[6] I. Molnar,“Goals, Design and Implementation of the New Ultra-Scalable O(1) Scheduler”, Linux Kernel, April 2002.
[7] Intel Corporation. Intel PRO/10GbE LR ServerAdapter,2003.

http://support.intel.com/support/network/adapter/pro10gbe/pro10gbelr/index.htm.
[8] J. D. Salehi, J. F. Kurose, and D. Towsley. The effectiveness of affinity-based scheduling in multiprocessor network protocol processing (extended version). IEEE/ACM Transactions on Networking, 4(4), 1996.
[9] Receive Side Scaling (RSS), http://www.microsoft.com/whdc/device/network/NDIS_RSS.mspx.
[10] Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms, Michael & Scott, Symposium on Principles of Distributed Computing, pp. 267-275, 1996
[11] Wang, J., Cheng, H., Hua, B., and Tang, X. 2009. Practice of parallelizing network applications on multi-core architectures. In Proceedings of the 23rd international Conference on Supercomputing, ICS '09. ACM.
[12] Intel Corporation, Supra-linear Packet Processing Performance with Intel Multi-core. http://www.intel.com/technology/advanced_comm/311566.htm, 2006.
[13] S. Kumar, et al., "Advanced Algorithms for Fast and Scalable Deep Packet Inspection", ANCS 2006.
[14] A. Mitra, W. Najjar and L. Bhuyan, "Compiling PCRE to FPGA for Accelerating SNORT IDS", ANCS 2007.
[15] F. Yu, et al., Fast and memory-efficient regular expression matching for deep packet inspection, ANCS 2006.
[16] Vik Desai, Microsoft Corporation, High-Performance Networking With NDIS 6.0, TCP Chimney Offload, and RSS., http://download.microsoft.com/download/5/b/9/5b97017b-e28a-4bae-ba48-174cf47d23cd/