Analysis of Mutual Exclusion Overhead of NDN Packet Forwarding on Multi-core Software Router

Junji Takemasa
Osaka University
Suita, Osaka, Japan
j-takemasa@ist.osaka-u.ac.jp

Yuki Koizumi
Osaka University
Suita, Osaka, Japan
ykoizumi@ist.osaka-u.ac.jp

Toru Hasegawa
Osaka University
Suita, Osaka, Japan
t-hasegawa@ist.osaka-u.ac.jp

ABSTRACT

This paper analyzes mutual exclusion overhead caused by Pending Interest Table (PIT) updates on multi-core Named Data Networking (NDN) software routers. The analysis reveals that an instruction to make the mutual exclusion atomic results in a bottleneck for high speed forwarding.

CCS CONCEPTS

• **Networks → Packet-switching networks**;

KEYWORDS

NDN, PIT, Mutual Exclusion

ACM Reference Format:
Junji Takemasa, Yuki Koizumi, and Toru Hasegawa. 2018. Analysis of Mutual Exclusion Overhead of NDN Packet Forwarding on Multi-core Software Router. In *5th ACM Conference on Information-Centric Networking (ICN '18), September 21–23, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3267955.3269016

1 INTRODUCTION

High speed forwarding on multi-core NDN software routers is an important research challenge. The challenge is being solved by eliminating high latency due to DRAM accesses [3, 4]. Most studies assume ideal sharding [2] such that incoming Interest and Data packets of the same name are allocated to the same thread in order to avoid mutual exclusion of updating their flow state. However, bursty incoming packets violate the assumption, and thus packets of the same name should be allocated to different threads, which raises an issue of mutual exclusion of such flow states. This paper analyzes mutual exclusion overhead on multi-core platforms.

2 MUTUAL EXCLUSION FOR PIT

2.1 Motivation

NDN stateful forwarding is pull-based communication where a state of an incoming Interest packet is saved until the corresponding Data packet arrives from an upstream router. Such a state is saved at a table called a PIT. An important feature of the communication is write-intensity such that a thread receiving an Interest or a Data packet always re-writes the PIT entry. When packets of the same name are allocated to different threads, accesses to the PIT entry of the same name should be mutually excluded. The motivation of this paper is to understand how mutual exclusion of PIT entries degrades NDN packet forwarding speed, whereas most existing studies keep away from this issue [3, 4].

ICN '18, September 21–23, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-5959-7/18/09.
https://doi.org/10.1145/3267955.3269016

2.2 Multi-Core Software Router Platform

We adopt a computer with recent multi-core CPUs as a router hardware platform. Each CPU has several cores, which are interconnected with a shared bus, and it has L1, L2, and L3 caches. While each core has exclusive L1 and L2 caches, all the cores share the L3 cache via the shared bus. An exclusive core is allocated to each thread to avoid context switches, which also results in significant overhead, and hence we interchangeably use cores and threads, hereafter.

2.3 Reference PIT Design

We employ a chained hash table proposed in [3] as a PIT data structure. It packs several pointers to hash entries, which are PIT entries of the previous subsection, on a hash bucket. The bucket is a cache line-sized array, so that this reduces the number of DRAM accesses. Each hash entry keeps a state of pending Interest packets, which records the name and the list of the incoming faces. A hash entry is updated every when an Interest or Data packet arrives, as described below: A thread fetches a hash bucket and then updates a list of incoming faces of the corresponding hash entry.

We use a lock mechanism to make a hash bucket update atomic. A lock variable, which indicates whether a lock of the corresponding hash bucket is released (0) or acquired (1), is implemented with the Compare-And-Swap (CAS) instruction.

2.4 Overhead Time of Lock Mechanism

This subsection identifies overhead time caused by the CAS instruction when the same hash bucket is shared with more than two threads. Figure 1 describes a scenario where the core #0 and the core #1 share a lock variable and the core #1 updated it. Thus, the lock variable is stored at the L1 cache of the core #1 and the state of the cache line which includes the lock variable becomes the *modified* state. At this time, the core #0 reads and updates the lock variable by executing the CAS instruction. Please note that we assume the MESI protocol [1] as a cache coherence protocol. An important feature of the CAS instruction is that the shared bus is locked to make the instruction atomic during its execution [1]. Hence, the other cores cannot access the L3 cache until the following steps are finished.

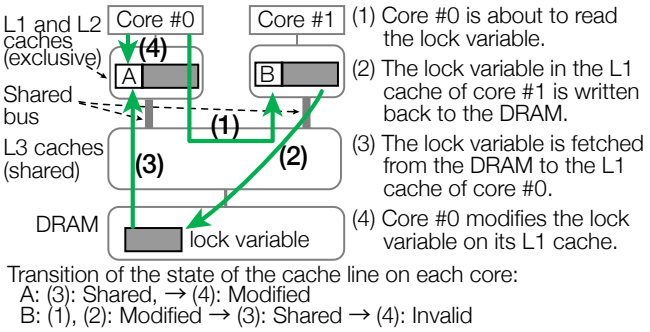

Figure 1: CAS instruction execution

(1) The CAS instruction at the core #0 locks the shared bus and then reads the lock variable at the L1 cache of the core #1 via the shared bus. (2) Since the cache line at the core #1 which stores the lock variable is in the modified state, the cache line is written back to the DRAM device. (3) The cache line in the DRAM device is transferred to the L1 cache of the core #0 and its state is set to the shared state. (4) The core #0 completes the CAS instruction, and thus the states of the cache line on the core #0 and the core #1 become the modified and the invalid states, respectively. The core #1 finally releases the shared bus.

An important observation is that the stale cache line of the lock variable incurs large time of accessing the DRAM device twice, as described in the aforementioned steps (2) and (3). If packets of the same name are allocated to different cores, the cache line of the core which handles the last packet is always stale because of the fact that a hash entry of the PIT is updated every time when an Interest or a Data packet is received.

3 ANALYSIS ON MUTUAL EXCLUSION

3.1 Analysis Method

In this section, we first analyze the number of CPU cycles spent by the CAS instruction empirically in the case that a PIT is designed naively such that a lock variable is locked as described in Section 2.3. Since it is expected that the cache line of the lock variable is in the modified state with high probability, two DRAM accesses occur. Second, we analyze the *ideal* and smallest CPU cycles spent by the CAS instruction in the case that the two DRAM accesses are eliminated. We analyze how the mutual exclusion of PIT updates prevents speedup of NDN packet forwarding with multiple cores.

We implement the above lock mechanisms on a router proposed in [4]. The router runs on a computer having one Xeon E5-2699 v4 CPU and DDR4 DRAM modules. To measure the number of CPU cycles of the naive implementation, the packets of 2^{20} names are randomly allocated to cores so that the CAS instruction accesses the DRAM twice with high probability. In contrast, the two DRAM accesses are eliminated by pre-fetching the cache line to the L1/L2 cache of the core which is locking it in the ideal implementation.

3.2 Analysis Results

The measured numbers of CPU cycles of the naive and the ideal implementations are 226 and 41, respectively, whereas the average

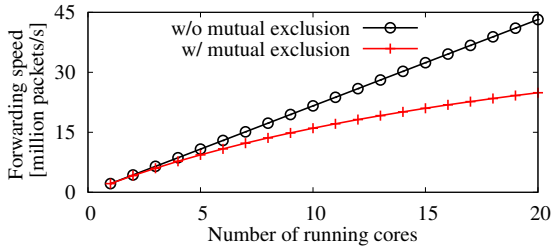

Figure 2: Forwarding speed of an NDN router with and without the mutual exclusion

number of CPU cycles for processing an NDN packet is 1018 on the same computer [4]. Pre-fetching the lock variable eliminates two DRAM accesses and the number of CPU cycles of CAS instruction accounts for about 4.1% of that of NDN packet processing. However, this overhead is a bottleneck for high speed NDN packet forwarding because the CAS instruction locks the shared bus, which prevents almost the all cores from running. That is, the CAS instruction results in sequential execution even on a multi-core platform.

Assuming an ideal case that the CAS instruction is executed sequentially but the other part of the NDN packet processing is able to be executed parallelly, we estimate the forwarding speed of the NDN router with and without the mutual exclusion of PIT updates. We use the number of CPU cycles spent by the ideal implementation of the mutual exclusion. Figure 2 shows the estimated forwarding speed when the number of cores varies from 1 to 20. The mutual exclusion decreases the forwarding speed by 42% compared with that of the NDN router without the mutual exclusion in the case that the number of cores is 20.

To mitigate mutual exclusion overhead, the probability of executing the CAS instruction should be minimized as well as lock variables should be pre-fetched in advance of executing the CAS instruction.

4 CONCLUSION

This paper analyzed the overhead caused by the mutual exclusion of the PIT and revealed that the CAS instruction results in a bottleneck for high speed forwarding because it stops all other threads regardless of whether they are about to access the same PIT entry.

ACKNOWLEDGMENTS

This work has been supported by the EU-Japan ICN2020 Project (EU HORIZON 2020 Grant Agreement No. 723014 and NICT Contract No. 184); and Grant-in-Aid for JSPS Fellows (17J07276).

REFERENCES

[1] Intel Corporation. 2018. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1. Retrieved July 23, 2018 from http://software.intel.com/sites/default/files/managed/7c/f1/253668-sdm-vol-3a.pdf
[2] Lorenzo Saino, Ioannis Psaras, and George Pavlou. 2016. Understanding Sharded Caching Systems. In *Proceedings of IEEE INFOCOM*. IEEE, 1–9.
[3] Won So, Ashok Narayanan, and David Oran. 2013. Named Data Networking on a Router: Fast and DoS-resistant Forwarding with Hash Tables. In *Proceedings of ACM/IEEE ANCS*. 215–226.
[4] Junji Takemasa, Yuki Koizumi, and Toru Hasegawa. 2017. Toward an Ideal NDN Router on a Commercial Off-the-shelf Computer. In *Proceedings of ACM ICN*. 43–53.