An FPGA NIC Based Hardware Caching for Blockchain

Yuma Sakakibara, Kohei Nakamura, and Hiroki Matsutani

Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
{yuma,nakamura,matutani}@arc.ics.keio.ac.jp

ABSTRACT

These days, people pay attention to Blockchain, which is a main technology of cryptocurrency. Blockchain is a fault-tolerant distributed ledger that does not need an administrator. We call transfer of digital asset as a "transaction". We need to hold all transactions to use Blockchain. Therefore, the amount of Blockchain data increases as time proceeds. On the other hand, the number of Internet of Things (IoT) products like smartphones has been increasing. It is difficult for IoT products to hold all Blockchain data because of their data capacity. Thus, they access Blockchain network via servers that have enough data capacity. More and more IoT products will join Blockchain network via servers, so it is useful to reduce workloads and improve throughput. In this paper, we propose caching technique using a Field Programmable Gate Array-based (FPGA) Network Interface Card (NIC) that has 10Gigabit Ethernet (10GbE) interfaces. More concretely, a Blockchain server receives a request from IoT products at the first time. Then, the server stores data on on-board DRAM of an FPGA NIC. Subsequently, FPGA NIC instead of the server responds to IoT products if the cache hits, so it can reduce server workloads. We implemented the proposed hardware cache to achieve high throughput on NetFPGA-10G board. We counted the number of requests that the Blockchain server processed at a given time and calculated throughput as evaluation. Throughput improved from 6.73 to 7.45 times when hitting the cache.

1. INTRODUCTION

Blockchain, which is a core technology of cryptocurrency, is a distributed ledger. It is also a digital payment system [1] using Peer-to-Peer (P2P) network which is fault-tolerant. Currently, financial business pay attention to Blockchain. For example, National Association of Securities Dealers Automated Quotations (NASDAQ) has developed "NASDAQ LINQ" [2], which is a new undisclosed stock distribution system to manage transactions effectively. In the near future, Blockchain will be more general system to be applied to various industry.

We call transfer of digital asset as a transaction. Blockchain makes a block by aggregating transactions at the interval of about 10 minutes. Networking nodes approve the block, and then it is connected to Blockchain. Blockchain is composed of all the transactions. Therefore, Blockchain data increases as time passes. For example, there are over 100GB of all the Blockchain data [3]. On the other hand, the number of IoT products [4] like smartphones has been growing. Generally, IoT products do not have enough data capacity compared to servers, so it is difficult for them to hold all the Blockchain data. To solve this problem, there is a system that servers hold all Blockchain data and IoT products receive a part of data in order to verify whether the transaction has already been approved by networking nodes or not [5]. In other words, the system enables IoT products access Blockchain via servers. However, the more the number of IoT products increases, the larger server workloads are needed. Therefore, it is important for servers to decrease their workloads.

In this paper, we propose hardware cache using an FPGA NIC in order to decrease server workloads. To be more specific, we design and implement key-value stores (KVS) hardware cache on an NetFPGA-10G board, which has a networking interface card with four 10GbE interfaces and an FPGA device. We evaluate throughput both when the cache hits and misses. As the result of evaluation, throughput improved by at most 7.45 times when hitting cache.

The rest of this paper is organized as follows. Section 2 introduces FPGA accelerators and issues of Blockchain. Section 3 illustrates our hardware cache using FPGA NIC. Section 4 evaluates throughput. Section 5 concludes this paper.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

2. BACKGROUND

2.1 Related Technology

2.1.1 Data Structure of Blockchain

Figure 1 shows the structure of Blockchain. Blockchain is a list of blocks, each of which is composed of a block header and a list of transactions [6]. The block header is made of three sets of block metadata. First, "Previous Block Hash" connects the block to the previous block in the Blockchain. Second, "Difficulty Target", "Timestamp" and "Nonce" are related data to the mining competition. Third, "Merkle root" is a data structure used to efficiently summarize all the transactions in the block [6]. A transaction is a data structure that shows a transfer of digital asset from a source to a destination. A transaction can be created by anyone and then signed with digital signatures. Then, the transaction is broadcasted on the network while it is validated by network nodes. Finally, the mining node verifies the transaction and makes a block [6]. It is important that the transaction is valid only when it is verified by a mining node and becomes part of Blockchain.

2.1.2 Merkle Tree

It is inefficient to verify all the transactions in the block in order to know whether the transaction has already been approved by networking nodes or not. Accordingly, there is a data structure called Merkle tree to verify the transaction efficiently. Figure 2 shows the structure of a Merkle tree. In Figure 2, transactions are expressed as Tx0, Tx1, Tx2, Tx3 and a hashed value of Tx0 is expressed as H_0. A Merkle tree is a binary hash tree that summarizes transactions of the block. A Merkle tree is made by connecting recursively hashed value

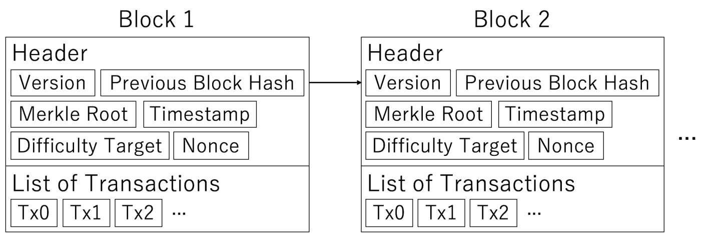
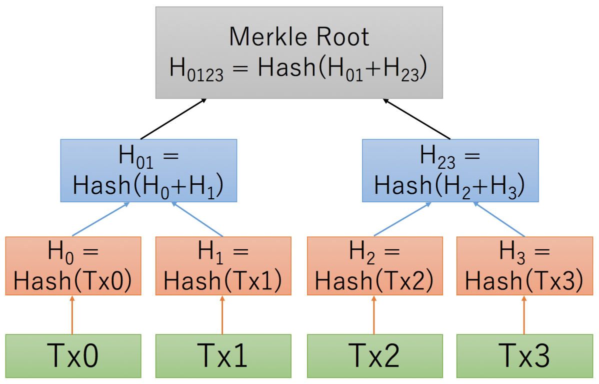


Figure 1: Structure of Blockchain



Merkle Root
H_{0123} = Hash(H_{01}+H_{23})
H_{01} = Hash(H_0+H_1)
H_{23} = Hash(H_2+H_3)
H_0 = Hash(Tx0)
H_1 = Hash(Tx1)
H_2 = Hash(Tx2)
H_3 = Hash(Tx3)
Tx0
Tx1
Tx2
Tx3

Figure 2: Structure of Merkle Tree

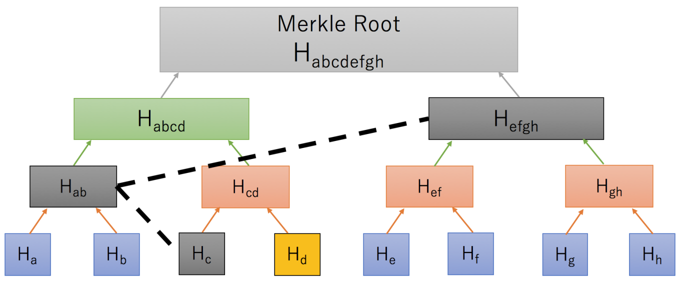


Figure 3: Producing Merkle Path

pairs of nodes until there is only one hash. The root of this tree is called a Merkle root. Even if there are thousands of transactions in a block, the size of Merkle root is always 32 Bytes as same as a transaction hash.

A node can prove whether a specific transaction is included in a block by producing $32log_2N$ Bytes hashed values if N transactions are included in a block. These hashed values which connect the specific transaction with the Merkle root are called Merkle path. In Figure 3, a node proves that a transaction d is included in a block by producing the Merkle path. The Merkle path consists of the three-hashed values H_c, H_{ab} and H_{efgh}. The broken line in Figure 3 indicates the Merkle path of the transaction d. There are only three-hashed values of $3 \times 32 = 96$ Bytes are needed although there are eight transactions of $8 \times 32 = 256$ Bytes in the block. Thus, a Merkle tree is efficient to prove that the specific transaction is included in the block.

2.1.3 *Types of Nodes*

The network of Blockchain is peer-to-peer network. Even though all nodes are equal in the peer-to-peer network, IoT products are unable to hold all the transactions. Therefore, nodes are classified depending on the functionality[6]. Functions are shown below.

- Wallet: a function of managing payment
- Miner: a funcntion of generating blocks by calculating hashes
- Blockchain Database: a function of managing all the transactions
- Routing: a function of participating in the network

A node which has all the functions above is called a Full node. Full nodes can verify all the transactions without any help of other nodes, but they require substantial computational resources. On the other hand, there is a node that does not require rich computational resources called a simplified payment verification (SPV) node [7]. The SPV node does not maintain all the transactions, but a block header. The chain of blocks, without transactions, is 1,000 times smaller than the full Blockchain [6]. Thus, IoT products such as smartphones, tablets and embedded systems can participate in Blockchain network as SPV nodes.

2.2 Related Work

We introduce previous work that is related to Blockchain and hardware cache using FPGA NIC.

There is previous work to achieve high throughput using KVS implemented on FPGAs. Memcached is a distributed in-memory KVS that improves response time of web servers by caching requested data on DRAMs in distributed servers. To improve energy efficiency of memcached, FPGA appliance has been proposed. As shown in [8], they proposed the design of memcached architecture implemented on FPGAs to achieve 10Gbps line rate processing. In [9], they proposed that the software memcached running on the host CPU by caching data and some operations at the FPGA NIC mounted on the server. In [10], they proposed a multilevel NoSQL cache architecture that utilized both the hardware cache of FPGA and software cache implemented in kernel.

Blockchain, a distributed ledger, achieved pseudonymous online payment, cheap remittance and digital asset exchange without an enormous central system [11]. On the other hand, Blockchain has scalability limits that trade-off between throughput and latency. The performance of Blockchain protocols is restricted by two parameters, block size and block interval. Increasing block size improves throughput, but the bigger blocks take longer to spread in the network [11]. Therefore, in [11], they proposed Bitcoin-NG, a scalable Blockchain protocol. The latency is limited only by the propagation delay of the network. Originally, Blockchain has been a core technology of cryptocurrecy without reliable financial institution. In addition to it, these days, a distributed application platform based on Blockchain is proposed. In [12], they proposed Ethereum, a transaction-based state machine. In [13], they proposed BigchainDB. BigchainDB is a decentralized database with Blockchain characteristics: decentralized control, immutability and creation or movement of digital assets. It combines the benefits of distributed databases and traditional Blockchain [13].

2.3 Summary

Traditional Blockchain realized digital asset exchange without an enormous central system to manage all the transactions. However, compared to other payment systems, the Blockchain protocol limited scalability and throughput. To make matters worse, increasing IoT products causes low throughput between a server and IoT products. To solve this problem, many previous researches have proposed design of new

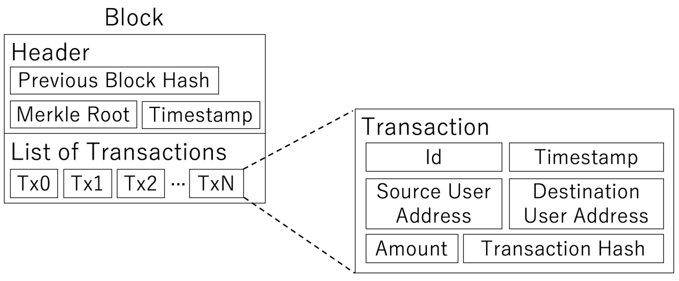


Figure 4: Structure of Simple Block

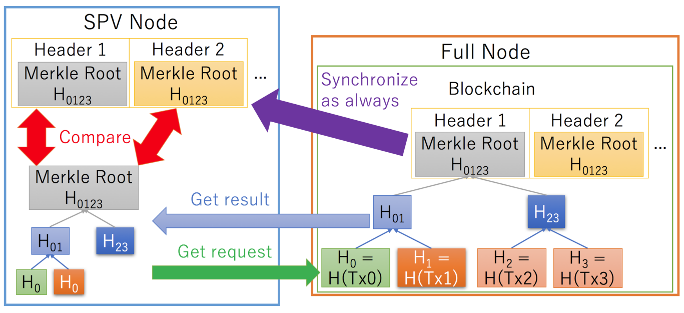


Figure 5: Verification of Transaction on SPV Node

Blockchain protocol. On the other hand, we propose KVS hardware cache to solve this problem.

3. HARDWARE CACHE ARCHITECTURE

3.1 Design

In this paper, we design and implement KVS hardware cache on FPGA NIC to respond promptly to SPV nodes when it verifies the transaction. SPV nodes do not possess all the transactions, so they cannot verify whether the transaction is included in the block or not by themselves. Thus, SPV nodes verify the transaction by accessing Blockchain via a Full node. In this section, we introduce structure of a simple block and how to verify a transaction on the SPV node, and then propose design and implementation of hardware cache.

3.1.1 Structure of Simple Block

For simplicity, we redesign structure of a block. Figure 4 illustrates structure of a simple block. The block header includes "Previous Block Hash", "Merkle Root" and "Timestamp". The transaction includes "Id", "Timestamp", "Source User Address" and "Destination User Address", "Amount" and "Transaction Hash". This structure can identify transaction from Source User Address, so the Full node can easily find out the transaction.

3.1.2 Verification of Transaction on SPV Node

Figure 5 shows verification of a transaction. In order to prove that the transaction has already been approved by other nodes and has become a part of Blockchain, an SPV node checks that the Merkle root made by the Merkle path corresponds with Merkle roots in block headers. We will explain how the SPV node verifies the transaction below.

Step 1: Receive block headers. An SPV Node receives block headers from a Full node at regular intervals, so the SPV node can extract Merkle roots from block headers (i.e. "Synchronize as always" in Figure 5).

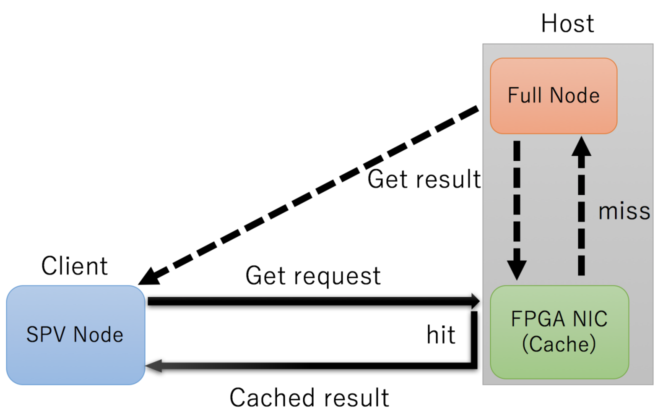


Figure 6: Proposed Hardware Cache System

Step 2: Send source user address to acquire the Merkle path. The SPV node sends the source user address to the Full node (i.e. "Get request" in Figure 5). The Full node receives the source user address and searches Blockchain for the latest transaction of the client based on the source user address. Then, the Full node makes the Merkle path corresponding to the transaction. After that, the Full node sends the Merkle path to the SPV node (i.e. "Get result" in Figure 5).

Step 3: Make and compare the Merkle root. The SPV node has the transaction hash. Thus, it can make the Merkle root by the transaction hash and the Merkle path by the Full node. Then, the SPV node compares the Merkle root made by the Merkle path with Merkle roots in block headers (i.e. "Compare" in Figure 5).

If one of Merkle roots in block headers is identical to the Merkle root made by the Merkle path, the transaction has already been approved. Therefore, the SPV node can provide service. Otherwise, the transaction has not yet been approved, so the SPV node requests again.

3.1.3 Get Request and Set Operation

We define a request and an operation of the Full node and the SPV node for hardware cache system.

- Get Request: The SPV node sends the source user address and then receives the Merkle path of client's latest transaction.

- Set Operation: The Full node searches Blockchain for the transaction based on source user address and make Merkle path of the transaction. After that, the Full node sends the Merkle path to the SPV node and then stores it on the FPGA NIC.

In this paper, the proposed KVS hardware cache stores "Source User Address" as the key and the Merkle path of client's latest transaction as the value. Figure 6 shows the proposed hardware cache system for Blockchain and the broken arrow indicates the behavior of cache system when cache misses while the bold arrow indicates the behavior of cache system when cache hits.

Figure 7 illustrates the behavior of cache system when cache misses while Figure 8 illustrates that of cache system when cache hits.

- Cache miss behavior

  An FPGA NIC receives the key and acquires the value from on-board DRAM. If a valid flag is equal to zero, the FPGA NIC does not cache requested Merkle path.

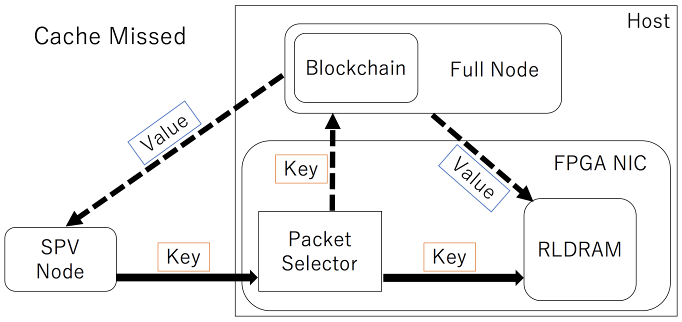


Figure 7: Behavior When Cache Misses

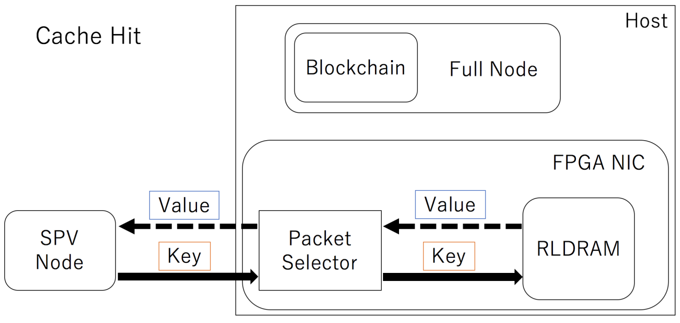


Figure 8: Behavior When Cache Hits

Therefore, it sends the key to the network protocol stack of a Full node. Then, the Full node searches Blockchain for the latest transaction of the client based on the key. After that, the Full node makes the Merkle path corresponding to the transaction and it replies the Merkle path (i.e. "Get result" in Figure 6) to the SPV node. After that, the Merkle path is cached on on-board DRAM of FPGA NIC.

- Cache hit behavior

  The FPGA NIC receives the key and acquires the value from on-board DRAM. If the valid flag is not equal to zero, it caches requested Merkle path. Thus, the FPGA NIC replies the Merkle path (i.e. "Cached result" in Figure 6) to the SPV node without the Full node. In this case, the workload of the Full node decreased, because FPGA NIC replies instead of the Full node.

3.2 Implementation

In Section 3.1, we designed KVS hardware cache in which the key is the source user address and the value is the Merkle path of user's latest transaction. In Section 3.2, we implemented KVS hardware cache in which the key is a 24-bit index generated from the source user address and the value is the Merkle root of user's latest transaction by considering the on-board DRAM size. Processing mechanism of FPGA NIC can be divided in three parts: payload extracting, accessing on-board DRAM of FPGA NIC and packet forwarding.

3.2.1 Payload Extracting

We employ User Datagram Protocol (UDP) as network protocol. Compared to Transmission Control Protocol (TCP), UDP is suitable for stream processing, because UDP is connectionless communication. Generally, payload in UDP packet is variable length. Payload of the UDP packet is the Merkle root of the client's latest transaction. In this paper, the length of UDP packets is 74 Bytes, because header length is 42 Bytes and payload is 32 Bytes. Advanced Extensible Interface (AXI) of NetFPGA-10G processes 32 Bytes in a cycle.

We assume that all the UDP packets whose destination port is pre-specified port number have keys and we call these packets as requesting packets. If an FPGA NIC receives a requesting packet, it extracts payload as the key. Otherwise, the FPGA NIC forwards packets to network protocol stack. To sum up, the FPGA distinguishes requesting packets from all the packets based on the destination UDP port. This process is called packet filtering.

3.2.2 Accessing On-Board DRAM of FPGA NIC

The Merkle root of the client's latest transaction, which is payload of the UDP packet, is corresponding to memory address of on-board DRAM. The FPGA needs to stop receiving new packets while accessing on-board DRAM, because the access latency is larger than the processing speed of FPGA. Therefore, the FPGA controls accessing on-board DRAM. The FPGA waits until on-board DRAM can be accessed. If it writes data on on-board DRAM, it validates write enable signal and sends write data and address to on-board DRAM interface. Memory address of on-board DRAM is generated by the source user address as a 24-bit index. If data is written to it, FPGA validates write done signal. On the other hand, if it reads data from on-board DRAM, it validates read enable signal and sends read address to on-board DRAM interface. If data is read from it, FPGA validates read done signal. The FPGA NIC receives next packet right after the process for the packet is done.

3.2.3 Packet Forwarding

The FPGA decides the node to send the packet depending on the value from on-board DRAM. If the valid flag is equal to zero, there is no requested data on on-board DRAM (i.e. cache misses). Thus, the received packet from the SPV node is forwarded to network protocol stack of the Full node. On the other hand, if the read data is not equal to it, requested data exists on on-board DRAM. Thus, the FPGA makes new UDP packet whose payload is the Merkle root of the client's latest transaction and return to the SPV node.

4. EXPERIMENTAL RESULTS

4.1 Evaluation Environment

4.1.1 FPGA NIC

In this paper, we employed NetFPGA-10G board [14], which possess four 10GbE interfaces and Virtex-5 FPGA, as a programmable NIC and cached data as key-value pairs. Figure 9 shows NetFPGA-10G board. Table 1 shows the specification of the NetFPGA-10G board. We implemented the hardware cache with Verilog HDL based on RLDRAM Stream provided by the NetFPGA-10G project [14] and used Xilinx ISE System Edition 13.4 for logic synthesis and place and route. We implemented 10GbE MAC with IP core provided by Xilinx and communication function provided by the NetFPGA-10G project [14].

Table 1: Specification of NetFPGA-10G

FPGA	Xilinx Virtex-5 (160MHz)
Optical Tranceiver	Four SFP+ interfaces (10Gb $\times$ 4)
Serial Interface	PCI Express Gen2 $\times$ 8

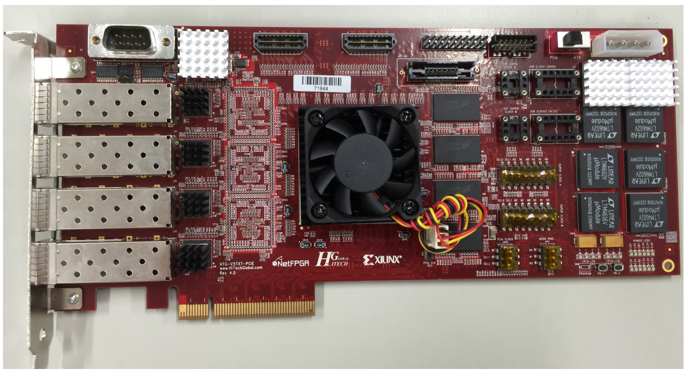


Figure 9: NetFPGA-10G Board

4.1.2 SPV Node and Full Node

Table 2 illustrates the specification of machines as SPV node and Full node. In order to emulate significantly huge numbers of SPV clients, we employ a single machine listed in Table 2.

Table 2: Specification of Machines

	SPV Node	Full Node
CPU	Intel Core i5-3470S	Intel Core i5-4460
Cores	4	4
Frequency	2.90GHz	3.20GHz
Memory	4GB	4GB
OS	CentOS 6.8	Ubuntu 16.04

4.1.3 Evaluation Method

The SPV client machine sends packets fully utilizing 10GbE bandwidth. We counted how many packets both the FPGA NIC and the Full node can process in a pre-specified interval and calculate throughput based on it. We employ the same NetFPGA-10G board both when cache hits and misses. We also employ NetFPGA Open Source Network Tester (OSNT) [15] as the packet sender in order to utilize fully 10GbE bandwidth. We made original C programs to implement SPV nodes and Full nodes.

4.2 Preliminary Evaluations

In this paper, we counted the number of packets both when cache hits and misses for evaluating our proposed hardware cache. We did preliminary evaluation beforehand in order to indicate that the number of SPV nodes and the number of transactions in a block hardly affect throughput in our simplified model, because these parameters are only a part of factors to determine the amount of total data. We did preliminary evaluations with our original C programs instead of OSNT.

4.2.1 Throughput When Number of SPV Nodes Changes

Figure 10 shows that the relationship between the number of all the transactions and throughput under the condition that the number of SPV nodes was changed to 5, 10 and 20. Figure 10 indicates that the number of SPV nodes scarcely affects the throughput.

4.2.2 Throughput When Number of Transactions in Block Changes

Figure 11 shows that the relationship between the number of transactions and throughput under the condition that the number of transactions in a block was changed to 1024, 2048 and 4096. The number of transactions in a block has been increasing. In [3], over 1,800 transactions are included in a block on average. Figure 11 indicates that the number of SPV nodes scarcely affects the throughput.

4.3 Hardware Cache Throughput

As the result of preliminary evaluation, throughput hardly depends on the number of SPV nodes and the number of transactions in a block, but depends on the number of all the transactions. Thus, we fix that the number of SPV nodes is 5 and the number of transactions in a block is 1024. Figure 12 shows that the relationship between the number of all the transactions and throughput both when cache hits and misses. In Figure 12, throughput improved from 6.73 to 7.45 times when the cache hits. We implemented software cache on the Full node in the evaluation. This is why throughput for cache miss is different from the max throughput in Figure 10 and Figure 11.

In this paper, we evaluated throughput both when cache hit and missed. We also considered the relationship between the hit rate of hardware cache and throughput. Assuming that H is the hit rate of hardware cache, T_0 is throughput when cache misses and T_{100} is throughput when cache hits, throughput T can be expressed as a linear equation below when the hit rate is H.

$$T = \frac{T_{100} - T_0}{100} H + T_0 \tag{1}$$

Throughput when cache misses depends on the number of all the transactions, so the number of all the transactions changes to 40,000, 240,000 and 480,000. In Figure 13, an error of throughput is small so as to be able to ignore it when cache misses.

5. SUMMARY

In this paper, we designed and implemented KVS hardware cache on on-board DRAM of FPGA NIC that has four 10GbE network interfaces in order to reduce server workloads. If cache hits, the FPGA NIC responses instead of the Full node. Thus, the Full node can reduce its workload and software overheads, because it does not need to access Blockchain database. Throughput between the Full node and IoT products can also be improved. We evaluated how the number of SPV nodes and the number of transactions in a block influenced throughput as preliminary evaluation. As the result, these two parameters hardly influenced throughput. Therefore, we fixed these parameters when we evaluate hardware cache throughput. For evaluation, the SPV node sent request packets to the Full node via FPGA NIC using fully 10GbE bandwidth. We counted the number of packets that both the Full node and FPGA NIC processed and calculated throughput based on it. As the result, throughput improved from 6.73 to 7.45 times when cache hit compared to when cache missed. Please note that this is the first work to address the server bottleneck of Blockchain which will be a crucial factor due to the increase of SPV nodes or IoT products in the near future.

In this paper, we used on-board DRAM to implement KVS hardware cache. The access latency of on-board DRAM is larger than the processing speed of the FPGA, so memory access can be bottleneck. Therefore, in the future, we will consider employing BRAM Block in the FPGA to cache data which are frequently used.

Acknowlegements This research presentation is supported in part by a research assistantship of a Grat-in-Aid to the Program for Leading Graduate School for "Science for Development of Super Mature Society" from Ministry of Education, Culture, Sport, Science, and Technology in Japan and in part by SECOM Science and Technology Foundation.

6. REFERENCES

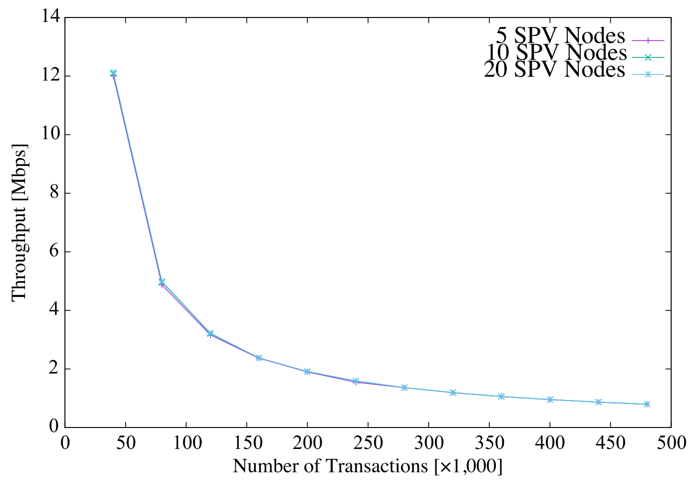


Figure 10: Throughput When Number of SPV Nodes Changes

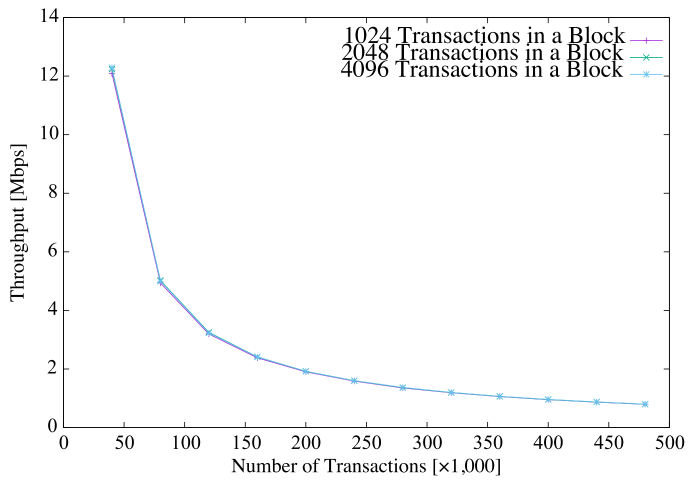


Figure 11: Throughput When Number of Transactions in Block Changes

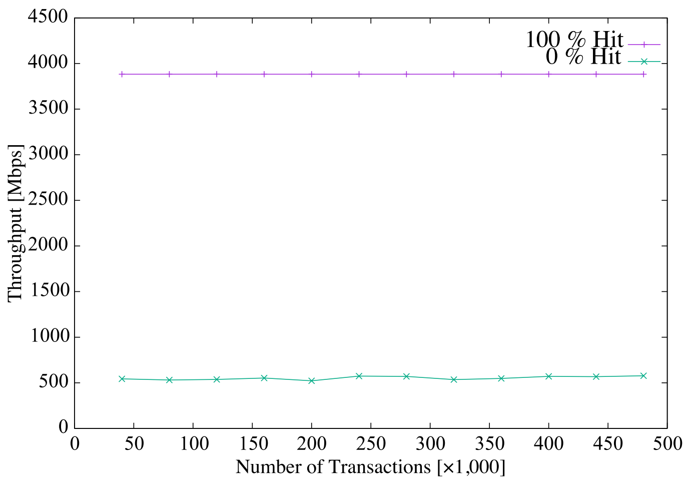


Figure 12: Relationship between Number of All Transactions and Throughput

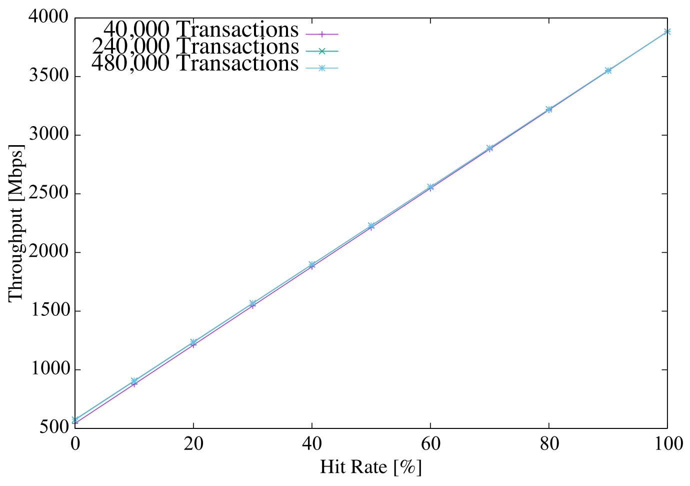


Figure 13: Relationship between Hit Rate of Hardware Cache and Throughput

[1] Kenji Saito and Hiroyuki Yamada. What's So Different about Blockchain?—Blockchain is a Probabilistic State Machine. In *Proceedings of the Distributed Computing Systems Workshops (ICDCSW'16)*, pp. 168–175, 2016.

[2] Sarah Underwood. Blockchain beyond bitcoin. *Communications of the ACM*, Vol. 59, No. 11, pp. 15–17, 2016.

[3] BLOCKCHAIN info. `https://blockchain.info/ja/charts/n-transactions-per-block#`.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, Vol. 54, No. 15, pp. 2787–2805, 2010.

[5] Adam Back, et al. Enabling blockchain innovations with pegged sidechains. `https://blockstream.com/sidechains.pdf`, 2014.

[6] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.

[7] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `http://bitcoin.org/bitcoin.pdf`, 2008.

[8] Michaela Blott, et al. Achieving 10Gbps Line-rate Key-value Stores with FPGAs. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'13)*, 2013.

[9] Eric S Fukuda, et al. Caching memcached at reconfigurable network interface. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'14)*, pp. 1–6, 2014.

[10] Yuta Tokusashi and Hiroki Matsutani. A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches. In *Proceedings of the High-Performance Interconnects (HOTI'16)*, pp. 60–67, 2016.

[11] Ittay Eyal, et al. Bitcoin-NG: A scalable blockchain protocol. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*, pp. 45–59, 2016.

[12] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014.

[13] Trent McConaghy, et al. BigchainDB: A Scalable Blockchain Database, 2016.

[14] The NetFPGA Project. `http://netfpga.org/`.

[15] Open Source Network Tester. `https://github.com/NetFPGA/OSNT-Public/wiki`.