IEICE TRANS. INF. & SYST., VOL.E101–D, NO.5 MAY 2018

PAPER

A Hardware-Based Caching System on FPGA NIC for Blockchain

Yuma SAKAKIBARA[†a)], ***Nonmember***, **Shin MORISHIMA**[†], ***Member***, **Kohei NAKAMURA**[†], ***Nonmember***, ***and*** **Hiroki MATSUTANI**[†], ***Member***

SUMMARY Engineers and researchers have recently paid attention to Blockchain. Blockchain is a fault-tolerant distributed ledger without administrators. Blockchain is originally derived from cryptocurrency, but it is possible to be applied to other industries. Transferring digital asset is called a transaction. Blockchain holds all transactions, so the total amount of Blockchain data will increase as time proceeds. On the other hand, the number of Internet of Things (IoT) products has been increasing. It is difficult for IoT products to hold all Blockchain data because of their storage capacity. Therefore, they access Blockchain data via servers that have Blockchain data. However, if a lot of IoT products access Blockchain network via servers, server overloads will occur. Thus, it is useful to reduce workloads and improve throughput. In this paper, we propose a caching technique using a Field Programmable Gate Array-based (FPGA) Network Interface Card (NIC) which possesses four 10Gigabit Ethernet (10GbE) interfaces. The proposed system can reduce server overloads, because the FPGA NIC instead of the server responds to requests from IoT products if cache hits. We implemented the proposed hardware cache to achieve high throughput on NetFPGA-10G board. We counted the number of requests that the server or the FPGA NIC processed as an evaluation. As a result, the throughput improved by on average 1.97 times when hitting the cache.
key words: *Blockchain, IoT, FPGA, cache*

1. Introduction

Blockchain is a distributed ledger which does not need an administrator to manage data. It has been originally a fault-tolerant digital payment system [1] using Peer-to-Peer (P2P) network. Therefore, Blockchain has initially been applied to financial business. 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. On the other hand, Blockchain is a fundamental concept to manage data dispersively. Thus, Blockchain will be applied to various industry in the near future.

Blockchain has many applications. In this paper, we mainly target Bitcoin-based Blockchain. In other words, we use Bitcoin as the example of Blockchain in the following explanation. A transaction indicates transfer of a digital asset in terms of Blockchain. Transactions are collected in a block. Blockchain is a chain of blocks, so Blockchain possesses all transactions. Blockchain size increases as time passes. In fact, Blockchain size is over 120GB of all [3].

Manuscript received September 13, 2017.
Manuscript revised December 27, 2017.
Manuscript publicized February 2, 2018.
[†]The authors are with the Graduate School of Science and Technology, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: yuma@arc.ics.keio.ac.jp
DOI: 10.1587/transinf.2017EDP7290

More and more people will possess Internet of Things (IoT) products to access the Internet. They will access Block network via IoT products. It is fact that the number of IoT products [4] like smartphones has been growing. On the other hand, IoT products generally do not have enough storage capacity compared to servers, so it is difficult for them to hold all Blockchain data. In order to solve the problem, there is a system in which IoT products manage Blockchain data via servers. Servers can possess all Blockchain data, so they send a piece of data to IoT products. IoT products verify whether the transaction has already been approved by networking nodes using the data [5]. However, if a lot of IoT products access servers at the same time, server workloads will significantly increase. Therefore, it is beneficial to decrease server workloads.

In this paper, we propose a hardware cache system using a Field Programmable Gate Array-based (FPGA) Network Interface Card (NIC) in order to decrease server workloads. We design and implement key-value stores (KVS) hardware cache on a NetFPGA-10G board with four 10GbE interfaces. The FPGA NIC instead of a server responds to requests from IoT products if cache hits. Therefore, the system can reduce server application latency and network latency. We evaluate throughput both when the cache hits and misses. As the result of evaluation, throughput improved by on average 1.97 times when hitting cache*.

Contributions of this paper are listed below.

- The proposed cache system can accelerate Blockchain applications even if it is accessed from a large number of IoT products.
- The proposed cache system reduces latency by 3.22 times and improves throughput by 1.97 times of a server.
- The proposed cache system can be used in Blockchain applications without modifying the protocol.

The rest of this paper is organized as follows. Section 2 explains Blockchain. Section 3 introduces previous work about FPGA accelerators and issues of Blockchain. Section 4 illustrates our hardware cache using FPGA NIC. Section 5 evaluates throughput. Section 6 concludes this paper.

*This paper is an extended version of our conference paper [6] by revising the description and adding new results.

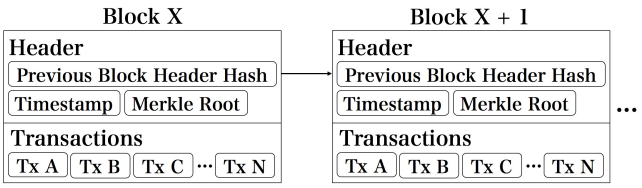

Fig. 1 Structure of Blockchain

Table 1 Data size of fields in block header

Field	Size
Previous Block Header Hash	32 Bytes
Timestamp	4 Bytes
Merkle Root	32 Bytes

2. Blockchain

Blockchain has unique features for IoT products. This section shows a system and features of Blockchain through data structure, Merkle Tree and nodes of Blockchain.

2.1 System

Any users can create a transaction with their digital signatures. Transactions are collected in a transaction pool. A node produces a block by gathering transactions at the interval of about 10 minutes with a lot of hash computation. It broadcasts the new block to the network and other networking nodes verify the block by computing hash of the block. If more than half nodes approve the block, it will be connected to Blockchain. The transaction becomes valid only when it is verified by network nodes and be part of Blockchain.

2.2 Data Structure

Figure 1 shows structure of Blockchain. Blockchain is a list of blocks, each of which is composed of a block header and a list of transactions [7]. The block header is made of three sets of block metadata. First, "Previous Block Header Hash" connects the block to the previous block in the Blockchain. Second, "Timestamp" is related data to the mining competition. Third, "Merkle Root" is a data structure used to efficiently summarize all transactions in the block. Table 1 shows data size of each field in a block header.

A transaction shows transfer of a digital asset from senders to recipients. Figure 2 shows simple structure of a transaction. In this paper, we simplify structure of transaction without loss of generality, because it includes essential data. A transaction is made of four sets of metadata, "TXID", "Timestamp", "INPUT" and "OUTPUT". "TXID" is Transaction ID which identifies the transaction. "Timestamp" indicates when a transaction is generated. "INPUT" includes "Previous Tx Hash" and "scriptSig" to refer to a previous transaction, which proves existence of asset. "OUTPUT" includes "Amount" and "scriptPubKey" to transfer asset to a recipient. Table 2 shows data size of each field in a transaction.

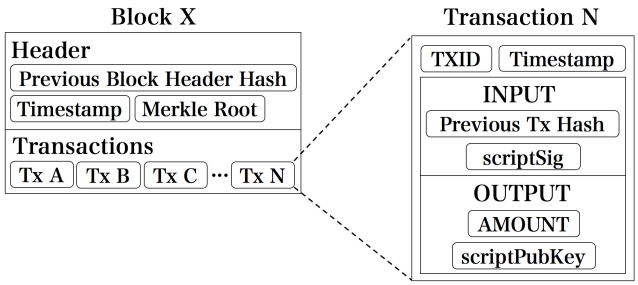

Fig. 2 Simple structure of transaction

Table 2 Data size of fields in transaction

Field	Size
TXID	32 Bytes
Timestamp	4 Bytes
INPUT	
Previous Tx Hash	32 Bytes
scriptSig	Variable Length
OUTPUT	
Amount	8 Bytes
scriptPubKey	Variable Length

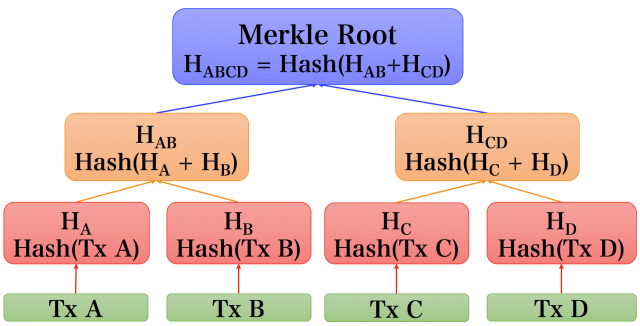

Fig. 3 Structure of merkle tree

2.3 Merkle Tree

It is inefficient to use all transactions in a block when verifying whether a transaction has already been approved by networking nodes. Accordingly, there is a data structure called Merkle Tree to verify the transaction efficiently. Figure 3 shows the structure of Merkle Tree. Merkle Tree is a binary hash tree that summarizes transactions of the block. Merkle Tree is produced by recursively connecting hashed value pairs of transactions until there is only one hash. In Fig. 3, transactions are expressed as Tx A, Tx B, Tx C, Tx D and a hashed value of Tx A is expressed as H_A. The root of Merkle Tree is called Merkle Root, which is expressed as H_{ABCD}. A hashed value never changes unless a transaction is tampered, so Merkle Root is used for verifying transactions. Even if there are thousands of transactions in a block, the size of Merkle Root is always 32 Bytes as same as a hashed value of a transaction. Therefore, Blockchain can save total

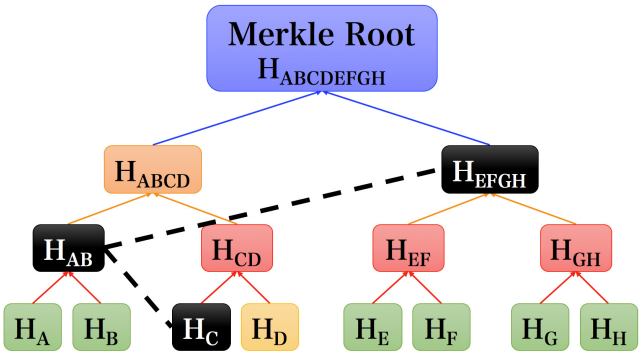

Fig. 4 Producing merkle path

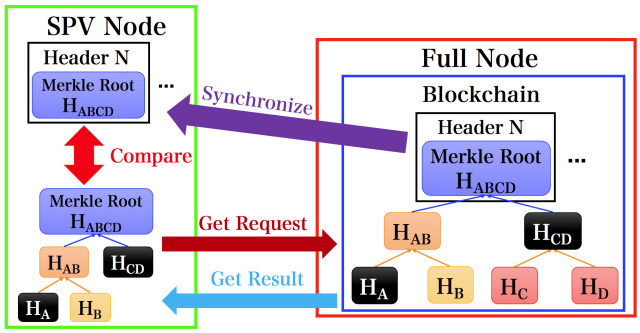

Fig. 5 Verification of transaction on SPV node

data size when verifying transactions.

A node, such as an IoT product, can prove whether a transaction is included in a block by using a part of Merkle Tree without possessing all transactions. To be more specific, the node can prove using hashed values of $32log_2N$ Bytes if N transactions are included in a block. These hashed values, which connect the specific transaction with Merkle Root are called Merkle Path. For example, a node proves that a transaction Tx D is included in a block by using Merkle Path in Fig. 4. Merkle Path consists of the three hashed-values H_C, H_{AB} and H_{EFGH}. The broken line indicates Merkle Path of the transaction Tx D. There are only three hashed-values of $3 \times 32 = 96$ Bytes are required although there are eight transactions in the block, which are $8 \times 32 = 256$ Bytes. Thus, Merkle Tree enables a node to prove efficiently whether a transaction is included in the block.

2.4 Types of Nodes

Blockchain is managed by Peer-to-Peer network. IoT products are unable to hold all transactions even though all nodes are ideally equal in the network. Therefore, nodes are classified depending on their functionality. 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 functions above is called a Full node. Full nodes possess all transactions, so they can verify transactions without any help of other nodes, but it is required for them to have substantial computational resources. On the other hand, there is a node, such as an IoT product, that does not have rich computational resources called a Simplified Payment Verification (SPV) node [8]. An SPV node possesses a block header and verifies transactions using Merkle Tree and Merkle Path. The chain of blocks, without transactions, is 1,000 times smaller than the full Blockchain. Thus, IoT products such as smartphones, tablets and embedded systems can participate in Blockchain network.

2.5 Verification of Transaction on SPV Node

SPV nodes must verify that a transaction is valid to provide any kind of service. Figure 5 shows verification of a transaction on an SPV node with Merkle Tree. An SPV node compares Merkle Root made by Merkle Path from a Full node with Merkle Roots in block headers to prove that the transaction has already been a part of Blockchain. The SPV node verifies the transaction in accordance with the following three steps.

Step 1: Synchronize block headers

An SPV Node receives block headers from a Full node as soon as possible, so the SPV node can extract Merkle Roots from block headers (i.e. “Synchronize” in Fig. 5).

Step 2: Acquire Merkle Path

The SPV node sends a sender address to the Full node (i.e. “Get Request” in Fig. 5). The Full node searches Blockchain for the transaction based on the sender address. Then, it makes Merkle Path corresponding to the transaction. After that, the Full node sends Merkle Path to the SPV node (i.e. “Get Result” in Fig. 5).

Step 3: Make and compare the Merkle Root

The SPV node has the transaction hash. Therefore, it can make Merkle Root by the transaction hash and Merkle Path receiving from the Full node. Then, it compares Merkle Root made by the SPV node with Merkle Roots in block headers (i.e. “Compare” in Fig. 5).

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

3. Related Work

This section introduces previous work that is related to hardware caching system using FPGA NIC and Blockchain.

3.1 Hardware Caching for Memcached

There are several pieces of previous studies to achieve high throughput using KVS caching system 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. FPGA appliance has been proposed in order to improve energy efficiency of memcached. As shown in [9], they proposed the design of memcached architecture implemented on FPGAs to achieve 10Gbps line rate processing. In [10], 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 to improve latency. In addition, FPGAs can be applied to NoSQL databases. In [11], they proposed a multi-level NoSQL cache architecture that utilized both the hardware cache of FPGA and software cache implemented in OS kernel. These studies showed the hardware cache using FPGA is efficient and effective to improve system performance.

3.2 Applications of Blockchain

Blockchain, a distributed ledger, achieved pseudonymous online payment, cheap remittance and digital asset exchange without an enormous central system. 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. Therefore, they proposed Bitcoin-NG [12], a scalable Blockchain protocol. The latency of Bitcoin-NG is limited only by the propagation delay of the network.

Originally, Blockchain has been a core technology of cryptocurrency. However, Blockchain is a fundamental concept that can be applied to other industry. A novel distributed application platform based on Blockchain is proposed these days. Ethereum [13], a transaction-based state machine, is proposed. Ethereum is a project which provides a trustful object messaging compute framework to end developers. BigchainDB [14], which combines the benefits of distributed databases and Blockchain, is proposed. BigchainDB is a decentralized database with Blockchain characteristics: decentralized control, immutability and creation or movement of digital assets. BigchainDB improved throughput of Blockchain by limiting public participation.

3.3 Our Contributions

Blockchain achieved digital asset exchange without a central system. However, compared to other systems, Blockchain protocol limited scalability and throughput. To make matters worse, increasing the number of IoT products will cause low throughput and high latency between a server and IoT products. In order to solve the problem, previous studies such as [12], [14] attempted to accelerate write performance of Blockchain applications by designing new protocols. However, in IoT era, read performance will be important as same as write one, because access from IoT products will increase. Our proposed hardware cache system can improve read performance of Blockchain applications without modifying protocol. Therefore, developers can apply our method and previous ones to Blockchain applications at the same time.

In addition, FPGAs were used for the hardware caching system with key-value stores in previous studies such as [10] and [11]. These systems were general, so their designs were not specific to Blockchain applications connecting with SPV nodes such as IoT products. On the other hand, we mainly focus on it. More concretely, Merkle Tree is inherent in Blockchain in order to make it easy to communicate with IoT products, so we considered caching Merkle Root. To sum up, we propose KVS hardware cache system on FPGA NIC to improve Blockchain system efficiently and effectively for when a lot of IoT products access simultaneously.

4. Hardware Cache Architecture

In this paper, we designed and implemented KVS hardware cache on the FPGA NIC. We show an overview of proposed KVS hardware cache architecture and then implementation of the architecture.

4.1 Design

In Sect. 2, we introduced types of nodes, an SPV node and a Full node for IoT products. We mainly focus on accelerating Blockchain connecting with IoT products, but the proposed system can be used more generally. Therefore, in the following, we use a term “client” as a node which receives a part of Blockchain data to verify transactions, “server” as a node which has Full Blockchain and “host” as composition of a server and an FPGA NIC. Please note that Blockchain is append-only, so consistency between Full Blockchain and cached data is maintained in the system.

4.1.1 Overview of Proposed System

Figure 6 shows behavior of the proposed KVS hardware cache system. The broken arrow indicates behavior when cache misses while the bold arrow indicates behavior of when cache hits. A client requests a host to get a required data (i.e. “Get” in Fig. 6). If cache hits, an FPGA NIC replies the data (i.e. “Cached Result” in Fig. 6) to the client. If cache misses, the request from the client is forwarded to the server (i.e. “Forward” in Fig. 6). After that, the server replies the data to the client (i.e. “Get Result” in Fig. 6) and saves data on the FPGA NIC (i.e. “Set” in Fig. 6). We will explain detailed cache behavior both when cache misses and cache hits below.

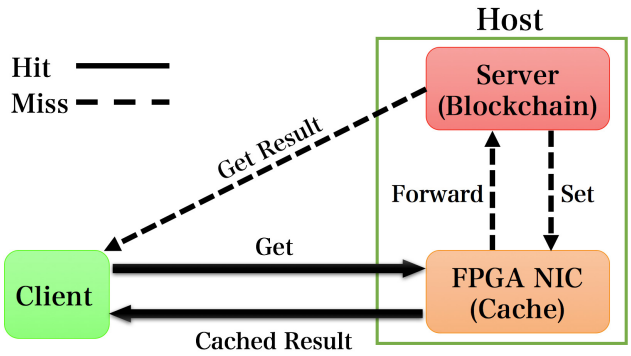

Fig. 6 Proposed hardware cache system

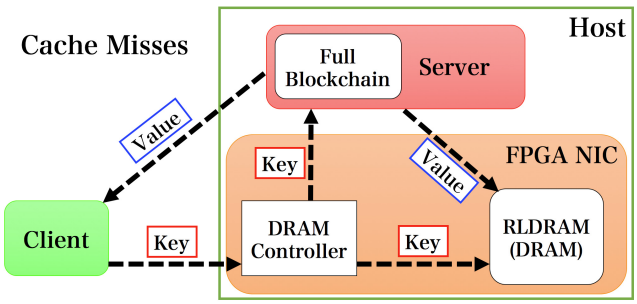

Fig. 7 Behavior when cache misses

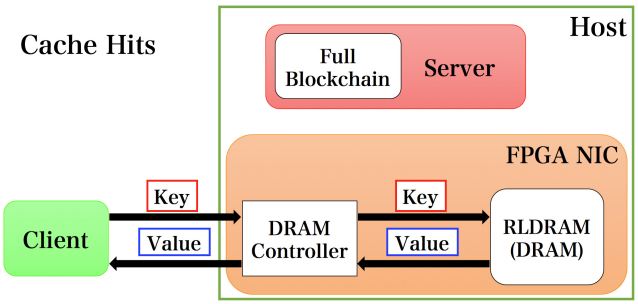

Fig. 8 Behavior when cache hits

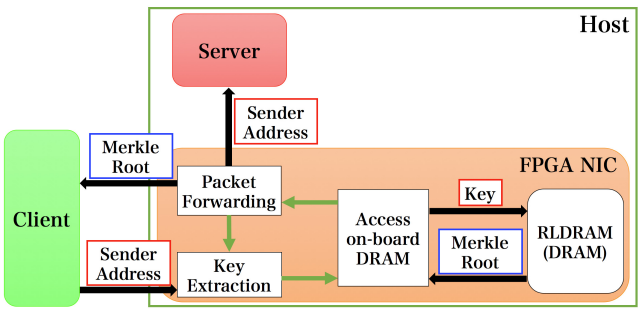

Fig. 9 Overview of hardware implementation

4.1.2 Cache Miss Behavior

Figure 7 illustrates behavior of cache system when cache misses. A client sends a key to a host to acquire a value. An FPGA NIC in a host receives the key at first and accesses on-board DRAM via DRAM controller. The value has not been cached on DRAM at the first time, so the FPGA NIC forwards the key to a server, which possesses Full Blockchain. Then, the server searches Blockchain for the value based on the key and it replies the value to the client. After that, the server accesses on-board DRAM of the FPGA NIC to cache key-value pair on it.

4.1.3 Cache Hit Behavior

Figure 8 illustrates behavior of cache system when cache hits. The client sends a key to a host, and the FPGA NIC receives the key and accesses on-board DRAM based on the key as same as when cache misses. It has cached the value at this time, so the FPGA NIC replies the value to the client without forwarding to the server. In this case, the workload of the server decreases, because the server does not process the request.

4.2 Hardware Implementation

We introduced an overview design of the proposed system, so we will explain its implementation on an FPGA NIC below. The proposed KVS cache system is built on Reduced Latency Dynamic Random Access Memory (RLDRAM) of the FPGA NIC, which is a type of DRAM. It is possible to implement the system on a typical DRAM in the future. The system is flexible, because data to be used as a key and a value can be selected depending on use cases. In this paper, we implemented an example of use cases. In Sect. 2, we showed simple structure of block and transaction. TXID identifies the transaction and Merkle Root is a summary of transactions in a block. Therefore, we assume the case when a client sends TXID to a host and receives Merkle Root to know whether the transaction is valid. In this case, the key is TXID and the value is Merkle Root of the block, which contains the transaction. The transport-layer protocol between the client and the server is User Datagram Protocol (UDP). We implemented the hardware cache based on the direct-mapped method. Line replacement occurs when same index is transferred to the same entry.

Figure 9 shows an overview of implementation. The FPGA NIC has three states, Key Extraction, Access on-board DRAM and Packet Forwarding, to manage the system.

4.2.1 Key Extraction

We assume that all UDP packets whose destination port is pre-specified port number have requested keys. We call these packets as requesting packets. The FPGA NIC extracts the payload, which is TXID, from the requesting packet and a lower 22-bit of TXID. It also operates DRAM controller to access on-board DRAM based on the key. Figure 10 shows request packet format. The length of requesting packets is

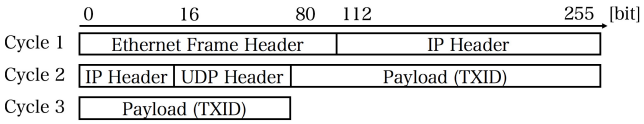

Fig. 10 Request packet format

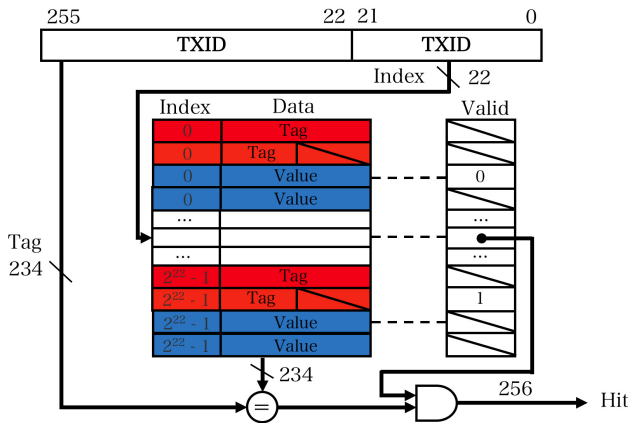

Fig. 11 Cache layout

74 Bytes, because header length is 42 Bytes and payload, which is TXID, is 32 Bytes. The AXI bus width is 32 Bytes, so the FPGA can extract 32 Bytes in a cycle. Therefore, it takes three cycles to extract payload from the packet whose size is 74 Bytes in this paper. If the FPGA NIC receives packets other than requesting packets, they are forwarded to the server.

4.2.2 Access On-Board DRAM

Figure 11 shows proposed cache data layout. According to NetFPGA-10G team [15], there are four RLDRAMs and their capacity is 288 MBytes in total. However, only 256 MBytes can be accessed from the board. More specifically, there are two memory controllers on the board, and each memory control is in charge of $2^{23} \times 128$ bits = 128 MBytes. In Fig. 11, "Tag" is marked as red while "Value" is marked as blue. A single cache line including "Tag" and "Value" uses up to 512 bits. Therefore, we fixed "Index" is a lower 22-bit of TXID and "Tag" is a higher 234-bit of TXID. "Valid" shows valid flags to judge whether cache hits or not. "Valid" is recorded in other registers, which correspond to "Index". "Value" is Merkle Root, which is 256 bits.

DRAM controller accesses on-board DRAM to judge whether cache hits or misses when it is received a requesting packet. We will explain how DRAM controller judges whether cache hits or misses below. It turns valid flags of on-board DRAM into disable at first. A valid flag is turned into enabled when the server saves key-value pair on the cache (i.e. "Set" in Fig. 6). Tag is used to check whether cached data is valid. If Tag is equal to a 234-bit value generated from the requesting packet and the valid flag is enable, DRAM controller judges as cache hits. Otherwise, it judges as cache misses.

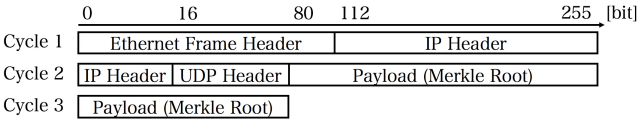

Fig. 12 Reply packet format

The FPGA NIC needs to stop receiving new packets while accessing on-board DRAM, because the access latency is larger than the processing speed of FPGA. The FPGA NIC waits until on-board DRAM can be accessed. Advanced Extensible Interface (AXI) of the FPGA NIC processes 32 Bytes in a cycle. Therefore, it takes more than a single cycle to acquire cached data. DRAM controller also accesses on-board DRAM when key-value pair which is provided by the server is cached.

4.2.3 Packet Forwarding

The FPGA NIC decides the destination to send the packet depending on whether cache hits or misses. If cache misses, the requesting packet from the client is forwarded to the server. On the other hand, if cache hits, the FPGA NIC makes a new UDP packet whose payload is Merkle Root to reply to the client by modifying the requesting packet. There are mainly two parts of the packet to be modified. First, the source MAC and IP addresses is switched for the destination MAC and IP addresses. Second, the payload is changed from TXID to Merkle Root. Figure 12 shows request packet format. The length of replying packets is 74 Bytes, because header length is 42 Bytes and payload, which is Merkle Root, is 32 Bytes.

4.2.4 Cache Organization

The hardware cache system is implemented based on the direct-mapped method, because it can mitigate cache complexity. Comparators in a set associative cache are required in response to the associativity, which causes performance degradation although it can reduce the conflict miss. We compared direct-mapped cache with 2 and 4-set associative cache memories by simulation from a view point of hit rate and number of tag comparisons to show the direct-mapped cache is enough. We used input data generated from bitcoind [16] to compare cache organizations. Bitcoind is a program which implements the Bitcoin protocol for remote procedure call. Figure 13 shows hash collision rates in different cache memories. We measured it by sending 10000, 50000 and 250000 keys. These keys are uniformly distributed, because they are hashed values of transations. According to Fig. 13, a hash collision rate of a directed-mapped cache is less than 1% when the number of keys is 250000, so negative impact of hash collision due to direct-mapped cache is limited. On the other hand, set-associative cache increases the design complexity. Therefore, in this paper, we adopted a direct-mapped method as cache organization.

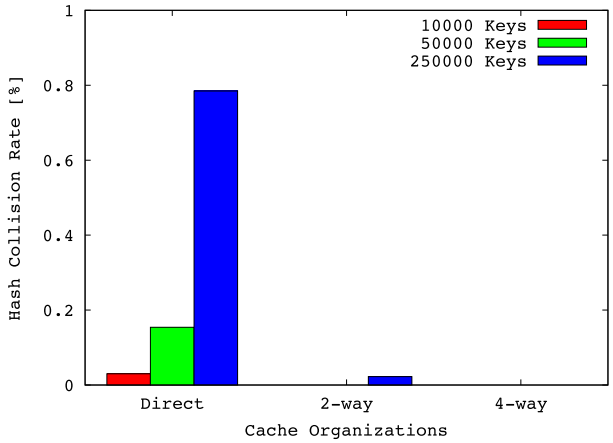

Fig. 13 Comparison of hash collision rates between caches

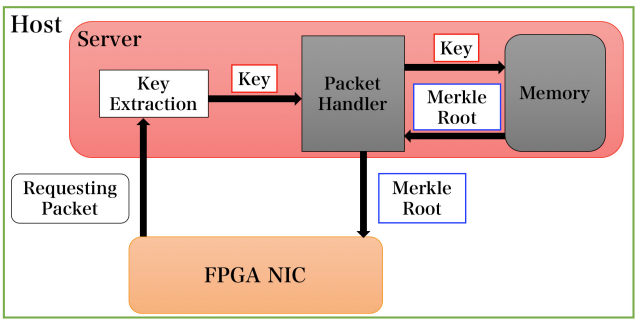

Fig. 14 Overview of handling packet on software implementation without cache

4.3 Software Implementation

We implemented two software application programs written by C language with Socket APIs. We adopted UDP as the transport layer. These programs were run on machines shown in Table 4. Main functions of software application programs are receiving packets from the FPGA NIC and retrieving corresponding value from prefixed Blockchain data on memory. Difference between two software application programs is whether implementing a software cache in order to measure throughput and latency caused by passing network and application processing. The software cache is implemented with a direct-mapped method and line replacement occurs when same index is transferred to the same entry. Therefore, basic structure of the software cache is same as the proposed hardware cache. Figure 14 shows an overview of handling packet on software implementation without cache while Figure 15 shows an overview of handling packet on software implementation with cache. The basic concept of handling packets on software implementations are same. Both software applications extract a key from the requesting packet. Then, a packet handler in a software application program without cache retrieves requested Merkle Root from the memory as shown in Fig. 14 and passes it to the FPGA NIC. On the other hand, a packet handler in another software application program acquires cached Merkle Root from the software cache as shown in Fig. 15 and passes it to the FPGA NIC.

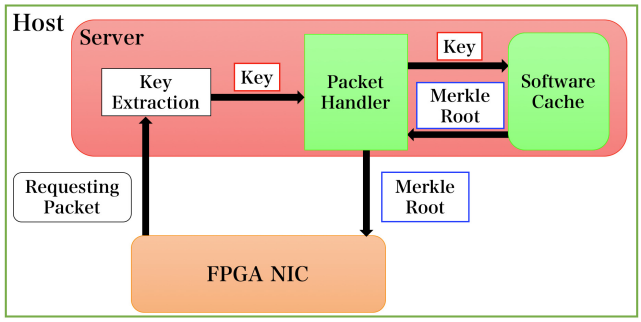

Fig. 15 Overview of handling packet on software implementation with cache

Table 3 Specification of NetFPGA-10G

FPGA Logic	Xilinx Virtex-5 TX240T (160MHz)
Networking Ports	Four SFP+ connectors (10Gb × 4)
DRAM	RLDRAM II (288MBytes)
Host Interface	PCI Express Gen1 × 8

5. Evaluations

We evaluated the proposed KVS cache system by measuring throughput. The proposed cache is flexible, because data to be used as a key and a value can be changed depending on applications. In this paper, as an example of using cache system, we cached TXID as a key and Merkle Root as a value.

5.1 Evaluation Environment

We used an FPGA NIC, a client and a server machine to evaluate the system.

5.1.1 FPGA NIC

We employed NetFPGA-10G board, which possesses four 10GbE interfaces and Virtex-5 FPGA, as an FPGA-based programmable NIC. We used a single 10GbE link out of four links. Table 3 shows the specification of the NetFPGA-10G board. We implemented the proposed KVS cache with Verilog HDL based on RLDRAM Stream provided by the NetFPGA-10G project [17] and 10GbE MAC with IP core provided by Xilinx and communication function. We also used Xilinx ISE System Edition 13.4 for logic synthesis and place and route.

5.1.2 Client and Server

Table 4 illustrates the specification of a client and a server machine. In this paper, we employed a single client machine listed in Table 4 in order to emulate significantly huge numbers of IoT products. We implemented original application

Table 4 Specification of machines

	Client	Server
CPU	Intel Core i5-4460	Intel Core i5-3470S
Cores	4	4
Frequency	3.20GHz	2.90GHz
Memory	8GB	4GB
OS	Ubuntu 16.04	CentOS 6.8
NIC	Mellanox ConnectX-3 EN 10GbE NIC	NetFPGA-10G Reference 10GbE NIC

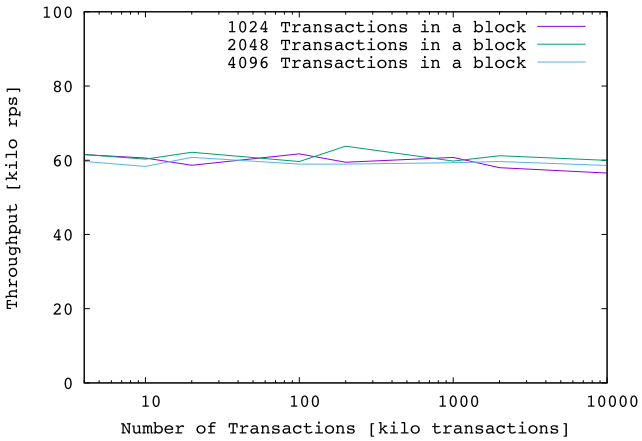

Fig. 16 Throughput when number of transactions in block changes

programs written in C language for the client and the server.

We counted the number of packets from the client machine that the FPGA NIC or the server could process and measured throughput as "requests per second (rps)". We used the same NetFPGA-10G board both when cache hits and misses. There are mainly one preliminary evaluation, throughput evaluations and area evaluation.

5.2 Preliminary Evaluation

The number of transactions in a block has been increasing. In fact, Blockchain Luxembourg [3] shows that over 1,700 transactions are included in a block on average. Therefore, we did preliminary evaluation beforehand in order to indicate that the total number of transactions rather than the number of transactions in a block affects throughput. We evaluated throughput when the number of transactions in a block was changed to 1024, 2048 and 4096. We search requested Merkle Root with a tree-based indexing. Figure 16 shows the relationship between the number of transactions and throughput.

Figure 16 indicates that the total number of transactions affects throughput rather than the number of transactions in a block. Therefore, we can fix that the number of transactions in a block is 2048 in other evaluations.

We argued software implementation in Sect. 4.3. Based on the measurement results, the number of cores used for software implementation is one and its utilization is 100%.

5.3 Throughput Evaluations

Figure 17 shows an overview of evaluation. We evaluated throughput while changing the number of blocks in

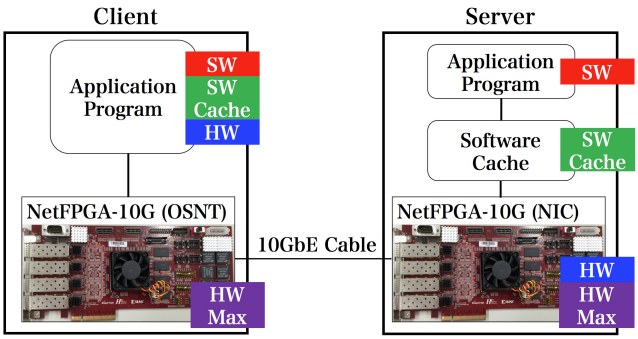

Fig. 17 Overview of four approaches

four different approaches: Software Approach, Software Cache Approach, Hardware Cache Approach and Hardware Cache Approach with NetFPGA Open Source Network Tester (OSNT) [18]. Workloads with the same key distribution condition are applied to four approaches.

1. Software Approach (i.e. "SW" in Fig. 17)
   The application program on the server processes requesting packets from client's application program. The application program searches requested Merkle Root with a tree-based indexing.
2. Software Cache Approach (i.e. "SW Cache" in Fig. 17)
   The software cache on the server processes requesting packets from client's application program. The basic structure of software cache is same as the proposed hardware cache, because both cache memories are implemented with a direct-mapped method and line replacement occurs when same index is transferred to the same entry.
3. Hardware Cache Approach (i.e. "HW" in Fig. 17)
   The proposed KVS hardware cache on the FPGA NIC processes requesting packets from client's application program. The basic structure of hardware cache is same as that of the software cache. The relation between throughput and hit rate will argue later.
4. Hardware Cache Approach with NetFPGA OSNT (i.e. "HW Max" in Fig. 17)
   The proposed KVS hardware cache on the FPGA NIC processes requesting packets from client's NetFPGA-10G with an OSNT hardware-based packet sender.

The proposed KVS cache system can reduce server application latency and network latency when cache hits. We also employed NetFPGA OSNT as a packet sender to fully utilize 10GbE bandwidth for only forth approach.

Figure 18 shows results of the four approaches. Four observations by comparing the four approaches are described below.

- Throughput improves by on average 1.07 times by reducing server application latency, comparing "SW" and "SW Cache".
- Throughput improves by on average 1.84 times by reducing network latency, comparing "SW Cache" and "HW".

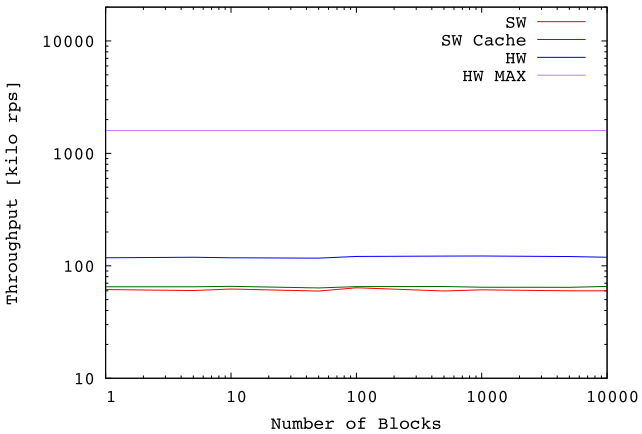

Fig. 18 Throughput in four approaches

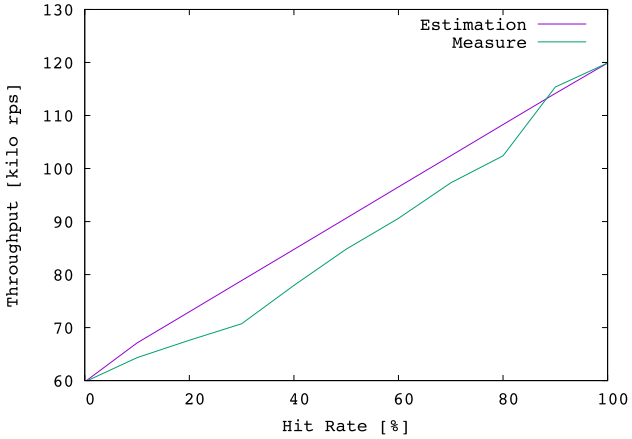

Fig. 19 Throughput while hit rate of hardware cache changes

- Throughput improves by on average 1.97 times by applying proposed KVS hardware cache, comparing "SW" and "HW".
- Throughput improves by at most 26.2 times under an ideal environment, comparing "SW" and "HW Max".

An ideal environment occurs when requesting packets are constantly sent and they fully fill 10GbE bandwidth.

We evaluated throughput both when cache hit and missed. However, throughput also changes depending on hit rate of the hardware cache in reality. The packet replies on the hit and miss performed simultaneously. Therefore, we consider the relationship between hit rate of the cache and throughput. We estimated throughput while hit rate changed. Assuming that H is hit rate of the cache, T_{SW} is throughput when cache misses and T_{HW} is throughput when cache hits, throughput T can be expressed as a linear equation below.

$$T = \frac{T_{HW} - T_{SW}}{100} H + T_{SW} \quad (1)$$

T_{SW} is measured throughput on average when the software application program without cache processes requesting packets, which is derived from "SW" in Fig. 18. T_{HW} is measured throughput on average when the proposed hardware cache processes requesting packets, which is derived from "HW" in Fig. 18.

We also measured throughput by changing hit rate of the cache. Figure 19 shows estimated throughput and measured throughput. As the result, an error of throughput is at most 10.4%.

5.4 Performance Model

We estimate performance model of the proposed method, which shows what the limitation of the proposed method is. There are two roofline models in Fig. 20: roofline with hardware and software. We use throughput in Fig. 18 to estimate roofline model with hardware and software. If applied workload increases, throughput also increases until it reaches the limitation of connection capacity. Performance of the proposed method is limited by 10GbE connection between an FPGA NIC and clients. Therefore, if the connection is fully used, this is the limitation of the proposed method.

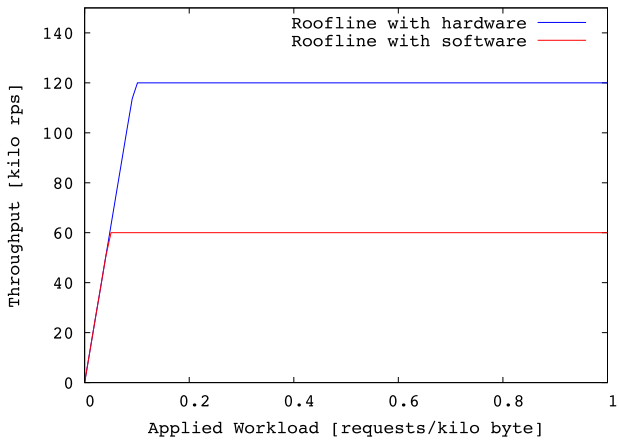

Fig. 20 Roofline model of proposed method

5.5 Latency Evaluations

This paper also proposed reducing server application latency and network latency, so we measured round-trip latency for four approaches. Latency includes network passing time for round-trip and system processing time. Please note that the latency and network bandwidth of Hardware Cache Approach and Hardware Cache Approach with NetFPGA OSNT is same, because the system of processing packets are same. Therefore, we describe "HW and HW MAX" in Fig. 21. Figure 21 shows results of the four approaches. Three observations by comparing the four approaches are described below.

- Latency improves by on average 1.64 times by reducing server application latency, comparing "SW" and "SW Cache".
- Latency improves by on average 1.97 times by reducing network latency, comparing "SW Cache" and "HW

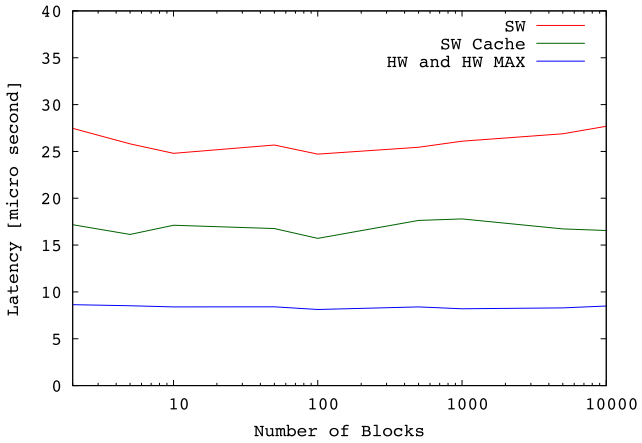

Fig. 21 Latency when number of transactions in block changes

and HW MAX".

- Latency improves by on average 3.22 times by applying proposed KVS hardware cache, comparing "SW" and "HW and HW MAX".

5.6 Area Evaluation

Table 5 shows an area evaluation of NetFPGA-10G board. Main hardware components actually implemented on FPGA are Reference NIC, a DRAM controller for RLDRAM and a hashing module to make addresses of RLDRAM. According to NetFPGA team [19], Reference NIC is mainly divided into five modules: input interface, input arbiter, output port lookup module, BRAM output queue and output interface. Packets first enter input interface and the input arbiter selects a packet from five input interfaces, four from the 10GbE interface modules and one from a DMA module, to the next stage. The output port lookup module decides which port a packet goes out of. After that, the packet is handed to the BRAM output queue until the destination is determined. Finally, output interface emits the packet to destination. Therefore, Reference NIC consumes all the FPGA hardware resources shown in Table 5. Function of a DRAM controller for RLDRAM is mainly described in Sect. 4.2 "Hardware Implementation". It controls RLDRAM for processing requesting packets. It consumes LUT, Memory, Slice Logic and Block RAM/FIFO to judge hit or miss. The hashing module to make addresses of RLDRAM is related to a DRAM controller. It generates a 22-bit hashed address of RLDRAM from a 256-bit TXID from clients. Therefore, the component consumes LUT, Memory, Slice Logic and Block RAM/FIFO. It indicates that utilization of Look-Up Table (LUT) and Block RAM (BRAM) is under 54%. The capacity of BRAM is 11,664 Kbits, so it is not employed as the hardware cache in this paper. However, the processing speed of BRAM is faster than that of RLDRAM. Therefore, as a future work, it is possible to accelerate further using the rest of BRAM for an alternative cache.

Table 5 Design summary

	Number (Total)	Utilization
LUT	44,250 (149,760)	29%
Memory	2,206 (39,360)	5%
Slice Logic	20,254 (37,440)	54%
IO	270 (680)	39%
Block RAM/FIFO	162 (324)	50%

Table 6 Power consumption

	Power [W]
SW	86.0
SW Cache	84.3
HW and HW MAX	67.5

5.7 Power Consumption Evaluations

We measured power consumption due to the proposed system. Table 6 shows the result of power consumption evaluations. Power consumption decreases 18.5W by introducing our proposed system, because the FPGA NIC processes requesting packets instead of a server. Software application programs on a server consumes more power than the proposed system on the FPGA NIC. Therefore, the proposed system is beneficial from the view point of power consumption.

6. Conclusions

The size of Blockchain, a distributed ledger, has been increasing. However, IoT products cannot possess all Blockchain data, so they access Blockchain via servers. If a lot of IoT products access servers at the same time, server workloads significantly increases. Therefore, in this paper, we designed and implemented KVS hardware cache using on-board DRAM of FPGA NIC in order to reduce server workloads. If cache hits, the FPGA NIC responses instead of the server. Thus, the server can reduce its workload and throughput between the server and IoT products can also be improved. We counted the number of packets that the server or the FPGA NIC could process and measured effective throughput. The proposed KVS cache system can reduce both server application latency and network latency. As the result, throughput mainly improves by on average 1.97 times applying proposed hardware cache system. In detail, reducing server application latency contributes on average 1.07 times throughput improvement while reducing network latency contributes on average 1.84 times throughput improvement. In reality, throughput depends on hit rate of the hardware cache. Therefore, we also estimated and measured throughput while changing hit rate. As the result, an error of throughput is at most 10.4%. 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 IoT products in the near future.

Acknowledgements

The authors thank Mr. Yuta Tokusashi for technical discussion.

References

[1] K. Saito and H. Yamada, "What's So Different about Blockchain? — Blockchain is a Probabilistic State Machine," Proc. Distributed Computing Systems Workshops (ICDCSW'16), pp.168–175, 2016.
[2] S. Underwood, "Blockchain beyond bitcoin," Commun. ACM, vol.59, no.11, pp.15–17, 2016.
[3] "BLOCKCHAIN info." https://blockchain.info/ja/charts/n-transactions-per-block
[4] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," Computer Networks, vol.54, no.15, pp.2787–2805, 2010.
[5] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timon, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," https://blockstream.com/sidechains.pdf, 2014.
[6] Y. Sakakibara, K. Nakamura, and H. Matsutani, "An FPGA NIC Based Hardware Caching for Blockchain," Proc. Highly-Efficient Accelerators and Reconfigurable Technologies (HEART'17), pp.1–6, 2017.
[7] A.M. Antonopoulos, Mastering Bitcoin: Unlocking digital cryptocurrencies, O'Reilly Media, Inc., 2014.
[8] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," http://bitcoin.org/bitcoin.pdf, 2008.
[9] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bar, and I. Zsolt, "Achieving 10Gbps Line-rate Key-value Stores with FPGAs," Proc. USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'13), 2013.
[10] E.S. Fukuda, H. Inoue, T. Takenaka, D. Kim, T. Sadahisa, T. Asai, and M. Motomura, "Caching memcached at reconfigurable network interface," Proc. International Conference on Field Programmable Logic and Applications (FPL'14), pp.1–6, 2014.
[11] Y. Tokusashi and H. Matsutani, "A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches," Proc. High-Performance Interconnects (HOTI'16), pp.60–67, 2016.
[12] I. Eyal, A. Efe Gencer, E. Gun Sirer, and R. van Renesse, "Bitcoin-NG: A scalable blockchain protocol," Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'16), pp.45–59, 2016.
[13] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," http://www.cryptopapers.net/papers/ethereum-yellowpaper.pdf, 2014.
[14] T. McConaghy, R. Marques, A. Muller, D.D. Jonghe, T.T. McConaghy, G. McMullen, R. Henderson, S. Bellomare, and A. Granzotto, "BigchainDB: A Scalable Blockchain Database," https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf, 2016.
[15] "NetFPGA-10G Information." http://netfpga.org/10G_specs.html
[16] "bitcoind." https://en.bitcoin.it/wiki/Bitcoind
[17] "The NetFPGA Project." http://netfpga.org/
[18] "Open Source Network Tester." https://github.com/NetFPGA/OSNT-Public/wiki
[19] "NetFPGA 10G Reference NIC." https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-Reference-NIC

Yuma Sakakibara is a Master's student at Keio University. His research interests include Blockchain and FPGA-based systems. He received a BE in computer science from Keio University in 2017.

Shin Morishima is a PhD student at Keio University. His research interests include graph database and GPU-based systems. He received an ME in computer science from Keio University in 2016.

Kohei Nakamura is a Master's student at Keio University. His research interests include Change Point Detection and FPGA-based systems. He received a BE in computer science from Keio University in 2016.

Hiroki Matsutani is an associate professor in the Department of Information and Computer Science at Keio University. His research interests include computer architecture. He received a PhD in engineering from Keio University in 2008. He is a member of IEICE, IPSJ, and IEEE.