2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications

Accelerating Blockchain Transfer System Using FPGA-Based NIC

Yuma Sakakibara, Yuta Tokusashi, Shin Morishima, and Hiroki Matsutani
Dept. of Information and Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku, Yokohama, Japan
Email: {yuma,tokusasi,morisima,matutani}@arc.ics.keio.ac.jp

***Abstract*—Blockchain is core technology for cryptocurrency and it is possible to become fundamental platform for industry and business. Especially, a blockchain-based digital asset transfer system using Internet of Things (IoT) products has recently been considered as a new practical platform, but the protocol limits performance. Previous research has improved performance by proposing new protocols, but further improvement is necessary for dealing with increasing transactions via IoT products. Therefore, we propose an in-Network Interface Card (in-NIC) processing approach using a Field Programmable Gate Array (FPGA) to improve performance of a blockchain-based transfer system. To be more concrete, we design and implement a prototype NIC with a key-value data store written in a P4 language on the FPGA that has four 10Gigabit Ethernet (10GbE) network interfaces. The prototype system supports frequently-used commands (CREATE, ISSUE, TRANSFER and REFER) for transferring digital asset. It reduces time for processing a kernel network protocol stack and accessing the data store. In fact, we measured throughput and latency of our prototype system compared to those of a blockchain software application. As a result, we found that our solution is able to obtain throughput 6.04 times higher on average and latency 15.4 times lower on average for all typical blockchain operations.**

***Index Terms*—Blockchain, FPGA, Key-Value Store, P4**

I. Introduction

Blockchain is one of the most noteworthy technologies in recent years. It is a fault-tolerant distributed ledger system using Peer-to-Peer (P2P) network. It has been a core technology for cryptocurrency such as bitcoin [1]. Currently, it provides a digital asset transfer system, which will be applied to various industries other than a financial application [2].

Transactions are a type of data structure that encode transferring of a value between participant nodes in a blockchain-based network system [3]. A specific node in the network aggregates transactions to create a block. The new block is validated and approved by other participating nodes. Afterwords, it is finally appended to the blockchain. Once a transaction is written to a blockchain, in principle, it will not be rewritten. The number of transactions using blockchain has been increasing significantly, because it is cheaper and faster than systems provided by traditional financial institutions. Actually, the total size of all block headers and transactions of bitcoin increase exponentially [4]. Furthermore, Internet of Things (IoT) products make users easy to access blockchain, so network traffic in blockchain will be congested due to increasing transaction volume via IoT products. On the other hand, a bitcoin-derived blockchain protocol has a limited throughput and latency [5]. Therefore, various platforms with new protocols have been proposed such as Ethereum and Hyperledger to improve them. However, these platforms are still not competitive with current database systems in term of throughput and latency in high workload scenarios [6]. Conventional research has been conducted based on protocol improvement, but further improvement is necessary to cope with increasing transaction volume. Therefore, it is important to process several transactions out of blockchain and final settlement of them is executed on blockchain, which is called an off-chain processing. Performance of an off-chain processing is not limited by a blockchain protocol, so we consider that throughput and latency of the off-chain processing will be required as same as those of a database system. In other words, we aim at 100,000 transactions per second (tps) as a required throughput and 7 micro seconds as a required latency.

In order to achieve high performance, we propose an in-Network Interface Card (in-NIC) processing approach on a Field Programmable Gate Array (FPGA) connected to a host machine to improve throughput and latency further. To be more specific, we design and implement a prototype system, which includes a key-value data store and supports fundamental commands for transferring a digital asset on an NetFPGA-SUME board, possessing four 10Gigabit Ethernet (10GbE) interfaces and an FPGA device. Hereinafter, the FPGA with NIC function is called an FPGA NIC. Actually, packet processing and data manipulation are executed inside the FPGA NIC without any software processing, so the prototype system can reduce processing time for a kernel network protocol stack and accessing time for the data store. We measured performance of the prototype system with four digital asset transfer commands and achieved the required throughput and latency. Our contributions are summarized as follows.

- We propose an in-NIC approach on the FPGA for improving throughput and latency of an off-chain processing out of blockchain.
- We design and implement a prototype system with a key-value data store, which supports four fundamental commands for transferring a digital asset on the NetFPGA-SUME board.
- An actual machine evaluation and simulation of the prototype system demonstrate the performance benefits.

978-1-7281-1141-4/18/$31.00 ©2018 IEEE
DOI 10.1109/BDCloud.2018.00037

The rest of this paper is organized as follows. Section II introduces related work about blockchain and FPGA and GPU-based accelerators. Section III illustrates a proposed blockchain-based transfer system using an FPGA NIC. Section IV describes implementation of the prototype system on the NetFPGA-SUME board. Section V shows experimental results. Section VI concludes this paper.

II. Related Work

Various approaches have been studied in order to improve performance of blockchain by modifying protocols. In terms of protocols, we classify types of blockchain for better understanding our target. Also, various approaches using FPGAs and GPUs have been studied for accelerating conventional systems. In this paper, we improve performance of a blockchain-based asset transfer system using an FPGA.

A. Blockchain Protocols

According to [7], in terms of fundamental properties, cost efficiency, performance and flexibility, blockchain can be classified into three types: public, private and consortium/community blockchain. Fundamental properties consist of five properties (immutability, non-repudiation, integrity, transparency, and equal rights). If a transaction is committed to a blockchain, data will eventually be immutable. A cryptographic technology provides a transaction for non-repudiation of the stored data and data integrity. Data transparency is provided by the public access. Participants have the equal ability to access and process blockchain, because it offers equal rights.

Public blockchain is adopted by most cryptocurrencies such as bitcoin, because it can be accessed by anyone on the Internet. It supports the fundamental properties, but sacrifices cost efficiency, performance and flexibility, so a public blockchain is not suited to business and industrial uses. On the other hand, generally, private blockchain can offer better cost efficiency and performance than public blockchain, because it limits members. A protocol of private blockchain does not fully support fundamental properties, so it is more vulnerable to malicious attacks than public blockchain. Usually, a trade-off exists between fundamental properties and performance. Even though write permission is held by a single organization, participant nodes within network must approve permission management. Consortium/community blockchain is used across multiple organizations. It mixes up features of public and private blockchain.

B. Blockchain Based Platforms

Most blockchain platforms have been implemented based on a public blockchain protocol, but currently some platforms for business use are built based on a private blockchain protocol. Ethereum and Hyperledger are major blockchain platforms for business use.

1) Ethereum: Ethereum [8] is an open-source public blockchain platform. It extends a bitcoin platform for business use by adding smart contract functionality which is a type of custom business logic. Generally, anyone can access an Ethereum network, but they can configure settings to be a private network. Participants manipulate data by applying contract codes. Therefore, they can transfer not only cryptocurrency but also develop their original assets by contract codes.

2) Hyperledger: Hyperledger is an open-source private blockchain platform developed by the Linux Foundation. Hyperledger Fabric [9] is one of the projects. It is designed to use for a cross-industry blockchain technologies, so it also offers smart contract functionality. Only permitted members can join the network, so it can attain better performance than Ethereum. Hyperledger Fabric has a world state [10], which holds the current values of a set of ledger states expressed in key-value pairs. Participants manipulate the state by executing a smart contract program called a chaincode. In other words, they can create their own original digital assets and transfer them by chaincodes.

A performance analysis of Ethereum and Hyperledger Fabric is conducted in [6] by constructing a cash transfer application. They implemented three functions (CreateAccount, IssueMoney and TransferMoney) to evaluate throughput and latency of the two platforms. They aimed for establishing a methodology for a blockchain-based platform and helping developers when adopting blockchain technology in their current systems. As the result, Hyperledger Fabric shows higher throughput and lower latency than Ethereum, but these platforms show an inferior performance to current database systems. Therefore, it is important to accelerate an off-chain processing in order to improve performance of current blockchain-based platforms.

C. FPGA and GPU Based Accelerators

FPGAs and GPUs are often applied to accelerate conventional systems and an application domain is varied. A key-value store (KVS) is often used for data management to achieve the high performance. Therefore, FPGAs and GPUs are suited to accelerate an off-chain processing. Previous work has proposed methods to accelerate a KVS by using FPGAs efficiently.

Memcached [11] is a general-purpose distributed memory cache system, which can accelerate response time of a website using a database by storing key-value pairs (KVPs) in distributed memories on servers. In [12], they employed FPGAs to accelerate memcached. They implemented main functionalities of memcached on an FPGA by integrating network, memory and compute interfaces. As the result, they improved energy efficiency compared to standard servers. In [13], they proposed the design of another memcached architecture implemented on FPGAs. They fully pipelined processing architecture, integrated network and memory interface by directly implementing on FPGAs and adopted modular design through standardized interfaces. As the result, they achieved 10Gbps line-rate KVS with FPGAs for all packet

sizes and they also improved latency and power consumption. In [14], they proposed a method for a low latency hardware memcached system with less memory. They stored the subset of data and several frequently-used commands at the NIC equipped with an FPGA. The NIC accesses Dynamic Random Access Memory (DRAM) to retrieve data when it receives a requesting packet from a client. If desired data is not cached in the DRAM, the NIC forwards the request to the host CPU running memcached. Consequently, they improved the latency of memcached.

FPGAs are also utilized for accelerating databases. Traditional database management system (DBMS) consumes the CPU resources for mission-critical transactional tasks. Therefore, in [15], they proposed a novel method of engine for database operations using an FPGA by offloading critical query operations using the FPGA. They implemented a prototype system using an FPGA and integrated it into commercial DBMS. As the result, they saved CPU resources and improved performance. In [16], we proposed a multilevel Not Only SQL (NOSQL) cache architecture. We used an FPGA-based hardware cache to complement an in-kernel software cache. We implemented a prototype system of the multilevel NOSQL cache on the NetFPGA-10G board and Linux Netfilter framework. As the result, we reduced the cache miss ratio and improved throughput.

Recently, a KVS is used for managing participant nodes of blockchain efficiently. In [17], we accelerated a blockchain-based system using an FPGA with a NIC function by caching merkle trees. We designed and implemented a cache on an FPGA-10G board in order to reduce servers' workloads and improve performance of the blockchain-based system. As the result, we improved performance. In [18], we also accelerated a blockchain-based system when searching full nodes using Graphics Processing Units (GPUs). We previously focused on accelerating functions of full nodes in a blockchain-based system by using FPGAs and GPUs. In this paper, we design and implement a KVS on an FPGA for accelerating an off-chain processing of a blockchain-based transfer system.

III. Proposed Blockchain-Based Transfer System

We simply design the prototype system for transferring a digital asset to achieve high performance. In other words, we integrate networking, computing and memory resources in an FPGA to accelerate a blockchain-based transfer system. Fig. 1 illustrates an overview of the proposed system. An FPGA NIC is connected to a host CPU by PCI Express (PCIe) to process a transaction packet complementarily. We design and implement core modules and a key-value data store in the FPGA NIC. Core modules process transaction packets from clients via network and issue one of the four digital asset transfer commands (CREATE, ISSUE, TRANSFER and REFER). The command is retrieved from the transaction packet and a module manipulates the key-value data store. After that, the core modules generate a reply packet with the result and send it to the client. The core modules do not process other packets such as verification packets, so they forward packets to the host CPU. It processes these packets according to a protocol and replies results to clients over the network.

Table I shows the four commands supported by the FPGA NIC. These are fundamental commands for a blockchain-based transfer system, so they are frequently requested by clients. CREATE, ISSUE and TRANSFER commands update the value while REFER command tells clients the current value. It is important to process these commands promptly in order to improve performance of the off-chain processing. An FPGA NIC can process these commands faster than a software application does. Therefore, we design and implement these functions on the FPGA NIC.

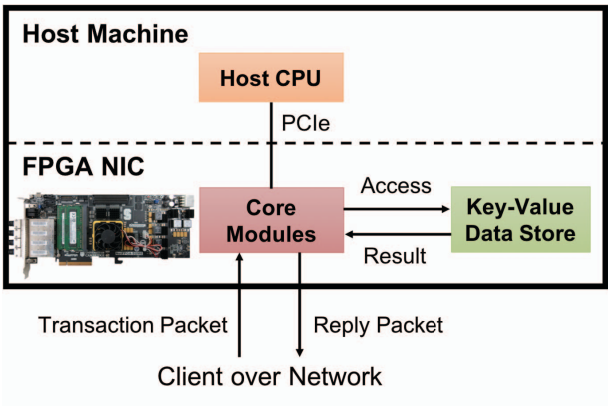

Fig. 1. Overview of Proposed System

TABLE I
Four Commands Supported by FPGA NIC

Command	Description
CREATE	Create a new key-value pair.
ISSUE	Issue a digital asset to a specified key by adding the specified value to the current value stored in the KVP.
TRANSFER	Transfer a digital asset from the specified key to the specified key by modifying the current values of a sender and a receiver stored in KVPs using a specified value.
REFER	Refer the current value stored in the KVP.

Fig. 2 illustrates contents of core modules in the proposed system. Networking, calculating and memory resources are aggregated into core modules. The core modules are mainly composed of four modules: packet selector, command extractor, data store controller and packet generator.

All packets from clients are collected in a packet selector module. It extracts a transaction packet from all packets. If a packet includes a transaction, the packet will be passed to the next module, the command extractor module. If not, the packet is forwarded to the host CPU via network interface.

The command extractor module chooses one of the four commands according to the description of the packet. If it is CREATE, the module extracts the key and passes it to a data store controller module. It accesses the key-value data store with the key to fix new key-value pair. The key is hashed by

a hash table when it accesses the data store. If it is ISSUE, the module extracts the specified key and a value from the packet and passes them to the data store controller module. It updates the current value in the data store by adding the value described in the packet. If it is TRANSFER, the module extracts three data. There are two keys for a sender and a receiver, and a transferring value. The command extractor module passes the data store controller module these data. It refers the sender's current value and subtracts the value from the current value once, in order to check whether the sender affords to transfer. If the sender does not have enough asset, the module returns an error message to the client. Otherwise, the module updates the data store by subtracting the value from the sender's current value and adding the value to the receiver's current value. If it is REFER, the module extracts the key and passes it to the data store controller module.

It accesses the data store to retrieve the current stored value. If the commands are CREATE, ISSUE or TRANSFER, the data store controller module sends the updated value to a packet generator module while if the command is REFER, it sends the referred value to a packet generator module.

The packet generator module generates a payload of a reply packet including the new value. It also generates a header of the reply packet by referring header information of the input packet from a client. It reverses source and destination MAC addresses, IP addresses and UDP ports. Finally, the reply packet is sent to the client via network interface.

Fig. 3 illustrates behavior of a key-value data store. The data store controller module accesses the data store with a hashed key. N represents the width of hashed key. The hashed key corresponds with an index of the data store. The data store has a valid flag to tell whether the corresponding key stores a valid value. Valid flags are managed by key value pairs by different modules. Valid flags are initially turned disable. A valid flag turns enable when the controller module executes a CREATE command. It checks a valid flag when executing the command. If the valid flag is enable, the controller module can update or refer the value of the data store. Otherwise, it cannot do it. Stored data will not be updated arbitrarily without proper command. Therefore, asset information of the client is protected safely.

IV. FPGA-Based Implementation

We implement the prototype system on the NetFPGA-SUME board. We describe implementation details in order of transaction data, data flow and experimental environment.

A. Transaction Data

Table II shows fundamental data constituting transaction. Transaction data of blockchain varies depending on a protocol, so we aggregate essential data into an original transaction for simple implementation and high performance. It is composed of Command, Transaction ID, Source ID, Destination ID and Amount, which are critical to execute transaction and update key-value data store. Although other data are also important, they are mainly used for transaction verification process.

TABLE II
Fundamental Data for Transactions

Data	Size	Description
Command	4Bytes	one of four commands
Transaction ID	4Bytes	serialized transaction ID
Source ID	4Bytes	ID for an asset sender
Destination ID	4Bytes	ID for an asset receiver
Amount	4Bytes	amount of transferring asset
Timestamp	8Bytes	timestamp of a transaction
UUID	16Bytes	universally unique identifier
Nonce	4Bytes	random number for cryptographic technology
Smart Contract Code	4Bytes	an application-level code for executing a transaction
Signature	Variable Length	digital signature for identification

B. Data Flow

Fig. 4 illustrates a data flow in FPGA NIC. Input packets from clients are sent to the FPGA NIC via network. We adopt User Datagram Protocol (UDP) and Internet Protocol version 4 (IPv4) as the network protocol, because it is suited to streaming data processing compared to Transmission Control Protocol (TCP). The packet format is as shown in Fig. 5. A payload includes Command, Transaction ID, Source ID, Destination ID and Amount. The rest of the payload is filled with the number 0 so that a packet size becomes 64Bytes. The FPGA NIC takes two cycles to receive the packet, because the NetFPGA-SUME receives and sends 256bits in a cycle using Advanced Extensible Interface (AXI) protocol. The input packet is initially passed through an input interface and an input arbiter chooses a packet from five input interfaces, four 10GbE Media Access Controller (MAC) modules and one Direct Memory Access (DMA) module, which is connected to the host machine. The packet is processed in the core modules as Fig. 2 and Fig. 3 described. If the command is CREATE, ISSUE or REFER, Source ID is hashed as the key when the controller accesses the key-value data store. If it is TRANSFER, each Source ID and Destination ID is hashed as the key respectively in order to access the data store. In this paper, N is fixed as 16 for the prototype system, but the size of N is variable depending on an FPGA or an application. The data store of the prototype system is implemented using Block RAM (BRAM) in the FPGA for attaining high performance. It is also reasonable to employ Dynamic RAM (DRAM) on the FPGA board for the data store for the protocol that does not limit participants. Core modules also decide which port a packet goes out of and the result packet is forwarded to an output queue module until the destination is fixed. Finally, an output interface emits the output packet to the destination.

C. Experimental Environment

Fig. 6 illustrates experimental overview. The prototype system works as the NIC on the server machine. We assume that requesting packets are aggregated into a single client machine, and it emits packets to the host machine sequentially. The client machine is connected to the server machine with 10GbE cable. An application program of the server machine

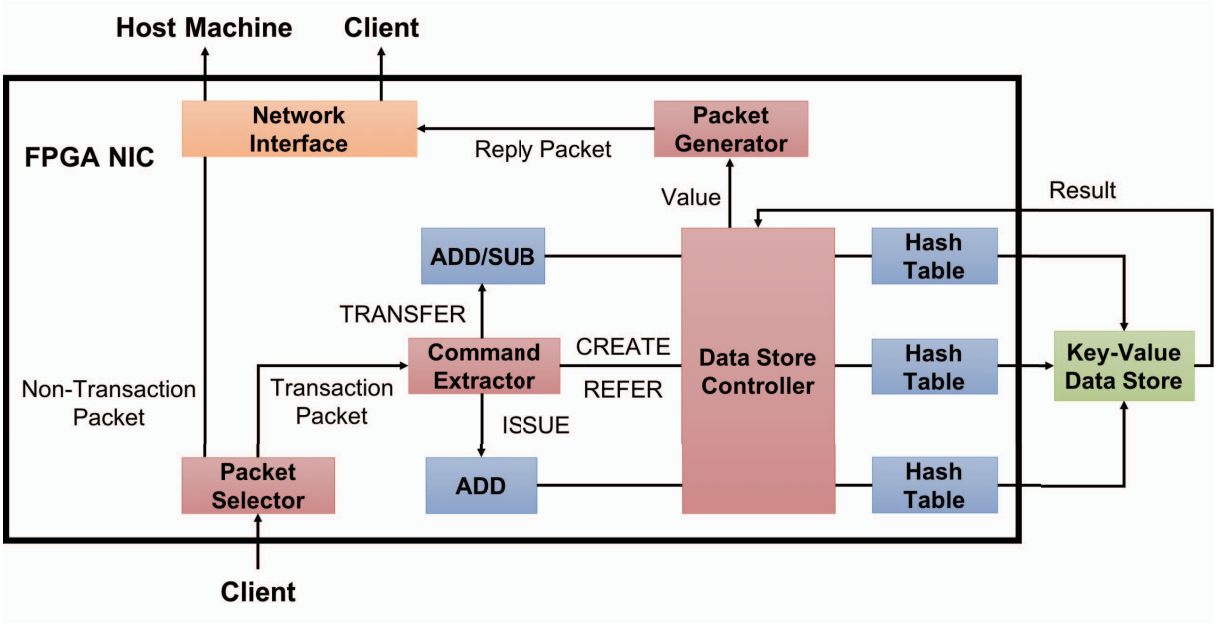

Fig. 2. Core Modules

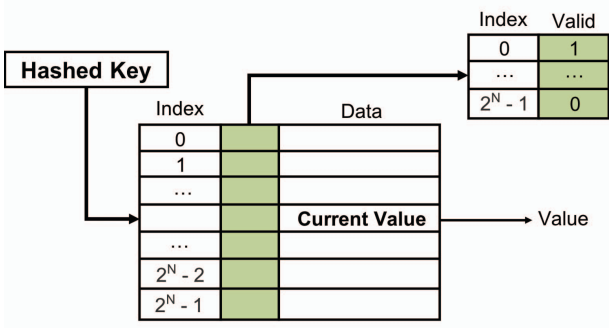

Fig. 3. Behavior of Key-Value Data Store

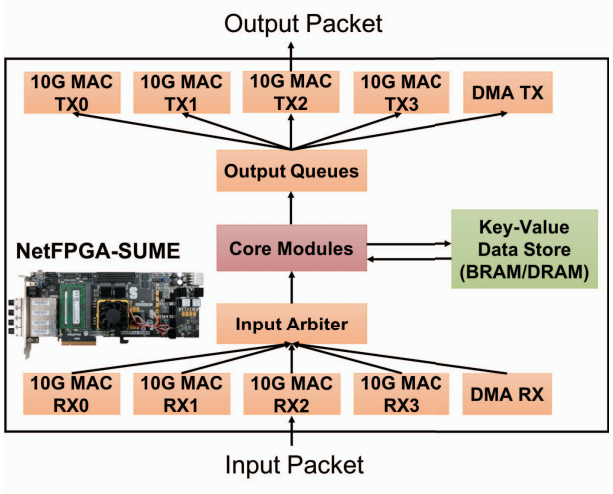

Fig. 4. Data Flow in FPGA NIC

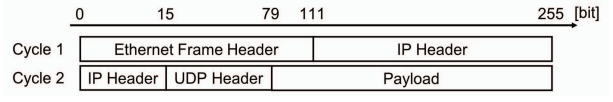

Fig. 5. Packet Format

is written in a C++ language, which has same functions with an FPGA NIC including a key-value data store. We used Tcpreplay [19], open-source utilities for emitting captured packets, as an application program of the client machine when evaluating throughput and latency. SW means software evaluation, which measures performance of the host CPU while HW means hardware evaluation, which measures performance of the FPGA NIC. We used same packets when we measured the performance of SW and HW.

Table III shows environment for client and server machines. We adopt NetFPGA-SUME board, which possesses four 10GbE interfaces and Virtex-7 FPGA, as an FPGA-based programmable NIC as Table IV shows. We used a single 10GbE link out of four links.

We use a P4 language, developed by open-source community P4 Language Consortium [20] for implementation. It is a domain-specific language for expressing network protocol. Currently, a P4 language has been paid attention to developers, because it enables them to reduce the time to implement new protocol [21]. A program written in P4 language is initially compiled by Xilinx Software Defined Specification Environment for Networking (SDNet) compiler [22] to yield

Register Transfer Level (RTL) architecture description. Xilinx Vivado design tools [23] secondly transform the RTL description to an optimized Xilinx FPGA implementation and finally generate the optimized bitstream for the Xilinx programmable device by integrating with IP cores provided by Xilinx. In other words, the tools execute logic synthesis and place and route. Maximum operating frequency of core modules is 202.35MHz, because a clock cycle is 5ns and worst negative slack (WNS) is 0.058ns.

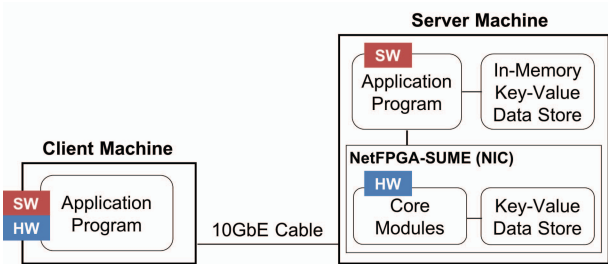

Fig. 6. Experimental Overview

TABLE III
ENVIRONMENT FOR CLIENT AND SERVER MACHINES

Item	Client Machine	Server Machine
CPU	Intel(R) Core(TM) i5-4460	Intel(R) Core(TM) i5-3470S
Operating Frequency	3.20GHz	2.90GHz
Memory	8GB	8GB
OS	Ubuntu 16.04	Ubuntu 16.04
NIC	Mellanox ConnectX-3 EN 10GbE NIC	NetFPGA-SUME Reference 10GbE NIC

TABLE IV
ENVIRONMENT FOR NETFPGA-SUME BOARD

Item	NetFPGA-SUME
FPGA	Virtex-7 XCV690T
BRAM	52,920Kb
DRAM	DDR3 SODIMM 4GB x2
SRAM	72MB QDRII+ SRAMs x3
PCIe	PCIe Gen3 x8
Network I/O	10GbE(SFP+) x4

V. EXPERIMENTAL RESULTS

We measured throughput and latency of the software application running on the host CPU and the prototype system running on the FPGA NIC when they execute the four commands (CREATE, ISSUE, TRANSFER and REFER). We counted the number of processed transactions per second (tps) as the throughput on condition that a packet includes a single transaction. We changed the number of packets to 1, 10, 100, 1000 and 10000 when measuring throughput. We simulated latency of core modules, excluding communication latency between the client and server machines, because it changes depending on an experimental environment. In addition, Xilinx Vivado Design Suite reports area utilization and power consumption.

A. Throughput

Figs. 7 to 10 show the throughput when the four commands are executed. The prototype system improves the throughput for all commands, compared to software application. It can process more packets than the software application, because it pipelines packet and data access processing. As the number of transactions increases, the throughput improvement rate also increases. This is because the amount that the FPGA can process in parallel is larger than that of the software application. Therefore, the results show that out proposed system is suitable for a blockchain-based transfer system, which is expected to increase transaction volume in the future. We summarize results of the four commands. Throughput improves 6.25 times on average when a CREATE command is executed. It improves 5.78 times on average when an ISSUE command is executed. It improves 5.97 times on average when a TRANSFER command is executed. It improves 6.14 times on average when a REFER command is executed. Totally, throughput of the prototype system improves 6.04 times on average, compared to that of the software application. We aimed at 100,000 tps as required off-chain processing performance reflecting from current database performance. The result satisfies required throughput when the number of transactions is over 100.

B. Latency

Fig. 11 shows the latency when the four commands are executed. Latency is measured under no load. The prototype system also improves latency for all commands, compared to the software approach. This is because it realizes lower execution latency of core modules at the key-value data store and reduces a processing time for passing a kernel network protocol stack. We summarize results of the four commands. Latency improves 17.2 times on average when a CREATE command is executed. It improves 17.5 times on average when an ISSUE command is executed. It improves 10.2 times on average when a TRANSFER command is executed. The TRANSFER command executes two operations. The controller module adds the value to a receiver's account and subtracts the value from a sender's account. Therefore, an improvement rate is lower than that of other commands. In addition, if the sender's current value is lower than the transferring value, the module returns an error message, so it also affects latency. It improves 16.6 times on average when REFER command is executed. Totally, execution latency of the prototype system improves 15.4 times on average, compared to that of the software application. We aimed at 7 micro second as required off-chain processing performance reflecting from current database performance. The result satisfies required latency for all commands.

C. Area Evaluation

Vivado Design Suite reports the FPGA NIC area utilization of core modules and the key-value data store. Table V shows area utilization of the FPGA referred from utilization design report. It consumes LUT, registers, BRAM, FIFO and I/O pins

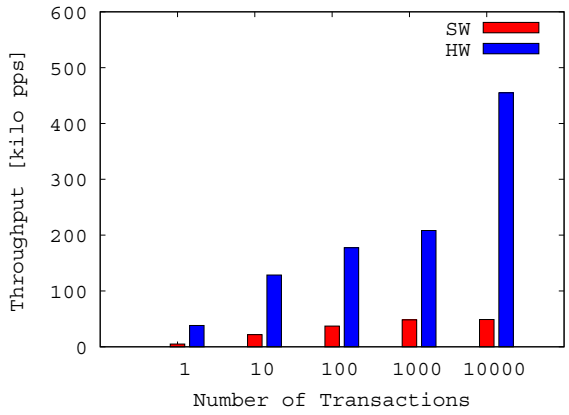

Fig. 7. Throughput for CREATE Command

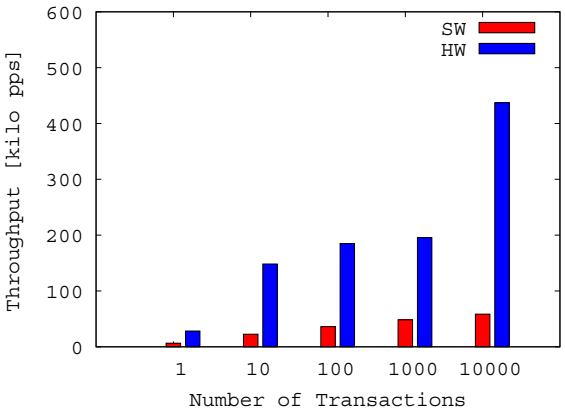

Fig. 8. Throughput for ISSUE Command

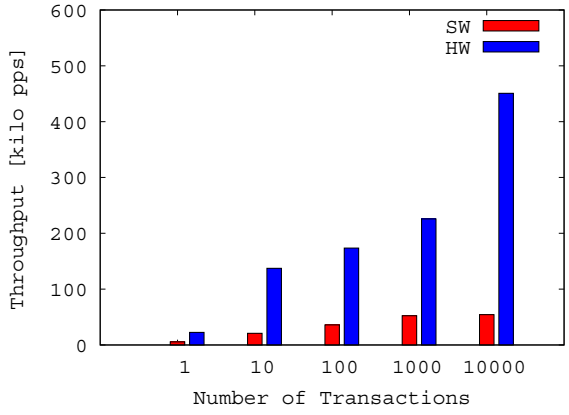

Fig. 9. Throughput for TRANSFER Command

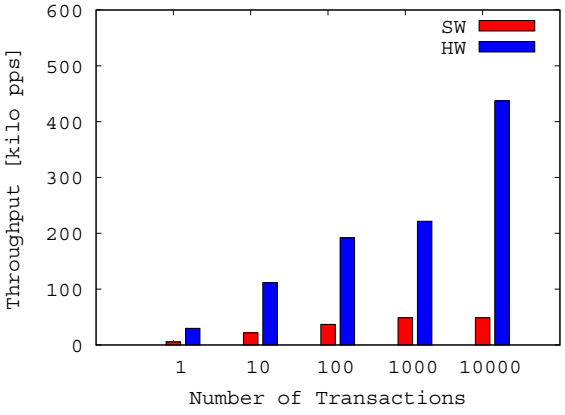

Fig. 10. Throughput for REFER Command

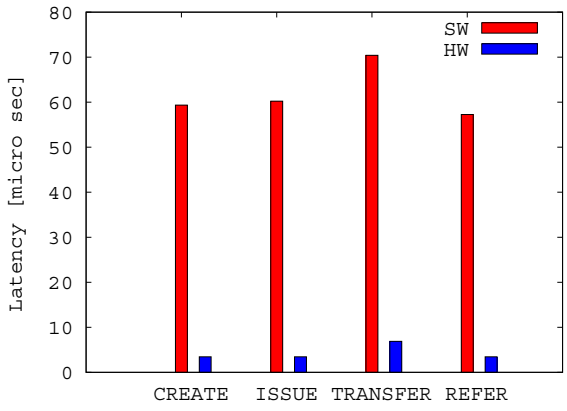

Fig. 11. Latency for CREATE, ISSUE, TRANSFER and REFER Command

for the prototype system. LUTs and registers are frequently used for processing the four commands and the hashing module. The key-value store is composed of BRAM, because we assume the number of users can be limited by the protocol. The resource utilization of BRAM whose capacity is 52,920Kbit, is 43.27%. It is considered to use DRAM to store larger data, but the access latency of BRAM is lower than that of DRAM. FIFO and I/O pins are used for packet transmission. A packet is waited in FIFO until the signal turns enable.

TABLE V
AREA UTILIZATION OF FPGA

Site Type	Number (Total)	Utilization [%]
LUT	129715 (433200)	29.94
Register	194887 (866400)	22.49
BRAM	636 (1470)	43.27
FIFO	596 (1470)	40.54
I/O pins	27 (850)	3.18

D. Power Consumption

Vivado Design Suite also analyzed power consumption of the prototype system. Table VI shows simulated power consumption of the FPGA referred from the power consumption report. Total on-chip power is 8.883W. The GTH transceivers

[24], which consumes 39 % of the total, are power efficient, and it is tightly integrated with the programmable logic resources. Clocks, signals, logics and BRAMs consume the power relatively high, because the key-value data store is used for all commands. On the other hand, MMCM, I/O and PCIE consume the power relatively low.

TABLE VI
POWER CONSUMPTION OF FPGA

Site Type		Power Consumption [W]	Rate [%]
GTH		3.508	39
Dynamic		4.876	55
	Clocks	1.265	26
	Signals	1.279	26
	Logic	0.814	17
	BRAM	1.022	21
	MMCM	0.321	7
	I/O	0.006	1
	PCIE	0.169	2
Static		0.498	6%

VI. SUMMARY

A blockchain-based transfer system has recently been paid attention for industrial uses, but the protocol limits performance. Previous research has focused on protocol modification to improve performance. However, further improvement for an off-chain processing not restricted by a protocol is necessary to cope with increasing transaction volume via IoT products. It has been reported that performance of software applications can be improved by constructing KVS on the FPGA. Therefore, we design and implement the prototype system with key-value data store on the NetFPGA-SUME board that has four 10GbE network interfaces. The prototype system supports fundamental commands (CREATE, ISSUE, TRANSFER and REFER) for a transferring digital asset. The system is expected to improve the performance of the system, because it reduces the processing time for passing a kernel network protocol stack and accessing the data store. We fixed 100,000 tps for throughput and 7 micro seconds for latency as required performance. In fact, we measured throughput and latency of the prototype system and the software application by requesting packets from a client machine. We also analyzed area utilization and power consumption of FPGA. As the result, our solution is able to obtain throughput 6.04 times higher on average and latency 15.4 times lower on average for the four commands. The results show that our system satisfies required performance and total on-chip power becomes 8.883W. In this paper, we employed BRAM instead of DRAM as the key-value data store and adopted UDP as the transfer protocol to achieve high performance. For future work, we would like to expand our implementation using DRAM and TCP in order to maintain consistency with existing systems.

ACKNOWLEDGMENT

This work was supported by JST CREST Grant Number JPMJCR1785, Japan.

REFERENCES

[1] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. http://bitcoin.org/bitcoin.pdf, 2008.
[2] 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)*, pages 168–175, June 2016.
[3] Andreas M Antonopoulos. *“Mastering Bitcoin”*. O’Reilly Media, Inc., 2014.
[4] Blockchain Luxembourg. https://blockchain.info/en/charts/n-transactions-per-block#.
[5] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol”. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI’16)*, pages 45–59, March 2016.
[6] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Thajchayapong Suttipong. “Performance Analysis of Private Blockchain Platforms in Varying Workloads”. In *Proceedings of the International Conference on Computer Communication and Networks (ICCCN’17)*, pages 1–6, July 2017.
[7] Xiwei Xu et al. “A Taxonomy of Blockchain-Based Systems for Architecture Design”. In *Proceedings of the International Conference on Software Architecture (ICSA’17)*, pages 243–252, April 2017.
[8] Gavin Wood. “Ethereum: A Secure Decentralised Generalised Transaction Ledger”. http://gavwood.com/paper.pdf, 2014.
[9] Elli Androulaki et al. “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”. In *Proceedings of the European Conference on Computer Systems (EuroSys’18)*, page 30, April 2018.
[10] Ledger - Hyperledger Fabric - Read the Docs. https://hyperledger-fabric.readthedocs.io/en/release-1.2/ledger/ledger.html.
[11] Memcached - a distributed memory object caching system. http://memcached.org/.
[12] Sai Rahul Chalamalasetti et al. “An FPGA Memcached Appliance”. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA’13)*, pages 245–254, Feburary 2013.
[13] 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)*, June 2013.
[14] 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)*, pages 1–6, September 2014.
[15] Bharat Sukhwani et al. “Database Analytics Acceleration using FPGAs”. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT’12)*, pages 411–420, September 2012.
[16] Yuta Tokusashi and Hiroki Matsutani. “A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches”. In *Proceedings of the IEEE International Symposium on High Performance Interconnects (HOTI’16)*, pages 60–67, August 2016.
[17] Yuma Sakakibara, Shin Morishima, Kohei Nakamura, and Hiroki Matsutani. “A Hardware-Based Caching System on FPGA NIC for Blockchain”. *IEICE Transactions on Information and Systems*, pages 1350–1360, May 2018.
[18] Shin Morishima and Hiroki Matsutani. “Accelerating Blockchain Search of Full Nodes Using GPUs”. In *Proceedings of the International Conference on Parallel, Distributed, and Network-Based Processing (PDP’18)*, March 2018.
[19] Tcpreplay - Pcap editing and replaying utilities. https://tcpreplay.appneta.com.
[20] P4 Language Consortium. https://p4.org.
[21] Marian Pritsak. “Is P4 Programming the Future of SDN?”. http://plvision.eu/p4-programming-future-sdn/, April 2018.
[22] Software Defined Specification Environment for Networking (SDNet). https://www.xilinx.com/publications/prod_mktg/sdnet/backgrounder.pdf, 2014.
[23] Xilinx Design Suite User Guide. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1118-vivado-creating-packaging-custom-ip.pdf, June 2018.
[24] 7 Series FPGAs GTX/GTH Transceivers. https://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf, August 2018.