An FPGA-Based Low-Latency Network Processing for Spark Streaming

Kohei Nakamura, Ami Hayashi, and Hiroki Matsutani
Dept. of ICS, Keio University
3-14-1 Hiyoshi, Kohoku, Yokohama, Japan 223-8522
Email: {nakamura,hayashi,matutani}@arc.ics.keio.ac.jp

***Abstract*—Low-latency stream data processing is a key enabler for on-line data analysis applications, such as detecting anomaly conditions and change points from stream data continuously generated from sensors and networking services. Existing stream processing frameworks are classified into micro-batch and one-at-a-time processing methodology. Apache Spark Streaming employs the micro-batch methodology, where data analysis is repeatedly performed for a series of data arrived during a short time period, called a micro batch. A rich set of data analysis libraries provided by Spark, such as machine learning and graph processing, can be applied for the micro batches. However, a drawback of the micro-batch processing methodology is a high latency for detecting anomaly conditions and change points. This is because data are accumulated in a micro batch (e.g., 1 sec length) and then data analysis is performed for the micro batch. In this paper, we propose to offload one-at-a-time methodology analysis functions on an FPGA-based 10Gbit Ethernet network interface card (FPGA NIC) in cooperation with Spark Streaming framework, in order to significantly reduce the processing latency and improve the processing throughput. We implemented word count and change-point detection applications on Spark Streaming with our FPGA NIC, where a one-at-a-time methodology analysis logic is implemented. Experiment results demonstrates that the word count throughput is improved by 22x and the change-point detection latency is reduced by 94.12% compared to the original Spark Streaming. Our approach can complement the existing micro-batch methodology data analysis framework with ultra low latency one-at-a-time methodology logic.**

I. Introduction

Stream processing is a processing paradigm that repeats an action of single or multiple operations on time-series data generated permanently, called stream data. There is a growing importance for processing and analyzing a large amount of stream data in low latency due to recent advances on sensing, IoT (Internet of Things), and SNS (Social Networking Service) technologies. For example, real-time trend business intelligence is required to collect, analyze, and perform various actions onto the stream data within a certain time period.

Existing stream processing frameworks are classified into micro-batch and one-at-a-time processing methodologies. In the one-at-a-time processing methodology, data analysis is performed whenever new data arrive along a data stream. In the micro-batch processing methodology, data analysis is periodically performed for a series of data arrived during a short time period, called a micro batch; thus, the stream data are discretely processed as multiple micro batches. Advantages of the micro-batch methodology is that existing software libraries for batch processing can be used for the micro-batch processing. Fault tolerance can be improved by creating checkpoints after processing micro batches.

Apache Spark [1] is one of large-scale data analysis frameworks that employ a batch processing methodology. Spark Streaming [2] is a stream processing library of Spark. Since it employs the micro-batch processing methodology, it can exploit rich libraries of Spark (e.g., machine learning, graph processing) for stream processing. However, an issue is that the micro-batch methodology inherently introduces a certain delay for processing depending on the micro-batch length.

Most stream processing frameworks including Spark and Spark Streaming are executed as a software program on microprocessors. Software program has a high flexibility, and we can speed up the data processing by various software optimization techniques. Please note that, when high bandwidth stream data (e.g., data stream received via 10Gbit Ethernet (10GbE)) are processed by an application program, all the received data are transferred from the network interface card (NIC) to an application layer via a TCP/IP network protocol stack. That is, in a conventional software-based stream data processing, all the data which may not be necessary for the applications are transferred to the application layer and then data processing tasks such as filtering and detection are performed. If we could perform data processing (e.g., outlier filtering and change-point detection) at the NIC, we can drastically reduce the data amount copied from the NIC to the application layer. Since data transfer between the NIC and the application layer via a kernel network protocol stack is a performance bottleneck [3][4], such in-NIC processing can improve the performance of stream processing.

In this paper, we propose to offload various one-at-a-time methodology operations onto FPGA (Field Programmable Gate Array) based 10GbE NIC and combine it with Spark Streaming to complement its negative aspect. More specifically, as an FPGA NIC, we employ NetFPGA-10G board [5] that has a Xilinx Virtex-5 FPGA device and four 10GbE interfaces and we implement the one-at-a-time methodology operations on the FPGA device. By performing the stream processing on the NIC, we can reduce data processing latency compared to the original micro-batch methodology Spark Streaming where a certain delay is inherently required depending on the length of each micro batch (e.g., 1 sec). In addition, by offloading the stream processing on the NIC and reducing (e.g., filtering) the data amount at the NIC, we can reduce the CPU workload and improve the throughput compared to the original Spark Streaming implemented as an application program.

The rest of this paper is organized as follows. Section II surveys stream processing frameworks and some FPGA-based accelerators. Section III introduces our approach and Section

IV illustrates our accelerator design using FPGA NIC for Spark Streaming. Section V evaluates it in terms of throughput and latency and Section VI conclude this paper.

II. Related Work

In this section, we introduce data processing frameworks and survey FPGA-based accelerations for stream processing.

A. Classification of Data Processing

Data processing can be classified into batch processing, one-at-a-time processing, and micro-batch processing [6]. We will introduce representative data processing frameworks listed in Table I.

TABLE I. Classification of data processing.

	Batch processing	One-at-a-time processing	Micro-batch processing
Spark [1]	Yes		
Storm [7]		Yes	Yes
Spark Streaming [2]	Yes		Yes

1) Spark: Batch Processing: Spark is a parallel distributed processing framework that has been developed in order to process a large amount of data at high speed. Different from Hadoop where intermediate data are stored in disks, Spark exploits in-memory data storages in order to mitigate the disk I/O overheads. Thus, it is suitable to iterative workloads (e.g., machine learning) that perform functions on the same data set repeatedly. In Spark, a data set is stored as an in-memory data storage called RDD (Resilient Distributed Dataset) [8] that consists of multiple partitions for distributed processing on a cluster of machines. For distributed processing, Spark first reads a data set from disks and stores it as an RDD. Then the RDD is distributed over multiple machines and processed by them based on the partitions. Fault tolerance can be improved by RDD [8]. Since a data set is stored as an RDD before processing, Spark is used for parallel distributed batch processing. In addition to Spark basic libraries that handle RDDs, various dedicated libraries are available, such as MLlib for machine learning, GraphX for graph processing, Spark SQL for SQL processing, and Spark Streaming for stream processing.

2) Spark Streaming: Micro-Batch Processing: Unlike the batch data, stream data are a series of data generated continuously over time. Unlike the batch processing that performs functions on stored data, stream processing performs functions (e.g., filter) on a series of data continuously generated. Spark Streaming is a library of Spark which is used to perform stream processing on Spark framework originally designed for batch processing. In Spark Streaming, a series of data is divided into small RDDs or micro batches at a regular time interval as shown in Figure 1. These micro batches are processed by utilizing Spark framework. By shortening the time interval of the micro batch, we can reduce the processing latency. Such micro-batch processing methodology strikes a good balance between stream processing capability and high productivity with rich libraries of Spark for batch processing. Micro-batch methodology improves fault tolerance by creating checkpoints at a regular time interval.

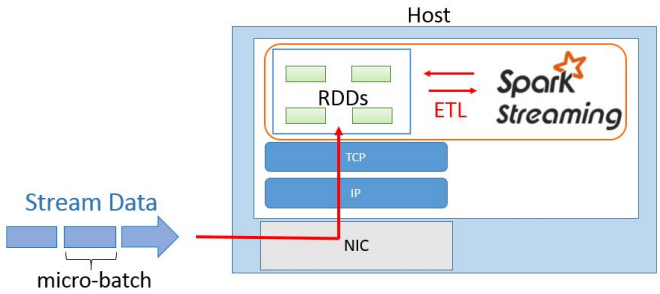

Fig. 1. Micro-batch processing of stream data.

3) Storm: One-at-a-Time Processing: Storm [7] is a distributed processing framework specialized for stream processing. In Storm, a pipeline processing of stream data is represented as a Directed Acyclic Graph (DAG) that consists of sources of data called Spouts and functions for the stream data called Bolts. Stream data is processed as a list of data called Tuple. Generation, distribution, and processing of Tuples are performed continuously so that stream data are processed in a pipeline manner. Bolt is performed for each Tuple as in a one-at-a-time processing methodology. Processing latency is shorter than those in batch and micro-batch processing methodologies, while computation becomes complicated and more compute resources are required for the one-at-a-time processing. Storm supports micro-batch processing methodology by utilizing a library called Trident.

B. FPGA-Based Acceleration for Stream Processing

FPGA-based accelerations have been extensively studied for stream processing. We will introduce some of them.

Machine learning is one of important tasks for stream processing. As machine learning algorithms are typically compute intensive, they have been offloaded to FPGA-based accelerators [9]. In [9], an outlier detection based on Mahalanobis distance was implemented on an FPGA NIC so that sensor data with normal values are discarded at the NIC and only those with anomaly values (i.e., outliers) are received by the host. By filtering unnecessary data at the NIC, computation cost including network protocol stack can be reduced. Change-point detection has been used for stream processing. Its hardware design was proposed in [10]. In [10], NP-CUSUM algorithm was implemented on an FPGA in order to demonstrate a high speed change-point detection. FPGA has a high parallelism, so it is possible to perform various algorithms at the same time.

A compiler and libraries for stream processing on FPGAs, called Glacier, have been developed in [11]. It translates a database query into hardware components described in VHDL. The hardware components are then implemented on FPGA to process stream data. Various operations for stream processing have been implemented on FPGAs. For example, join is one of basic operations in databases, and handshake join is a typical stream join algorithm. Stream join for FPGAs was studied in [12]. Hardware-friendly designs of the stream join and their scalability were discussed. Sliding window aggregation is also an important operation for stream processing. It can be used to detect anomalous conditions by monitoring real-time stream data. Since it requires a high computation power, its efficient

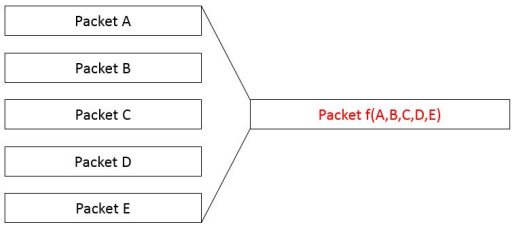

Fig. 2. Data size reduction by aggregate operation.

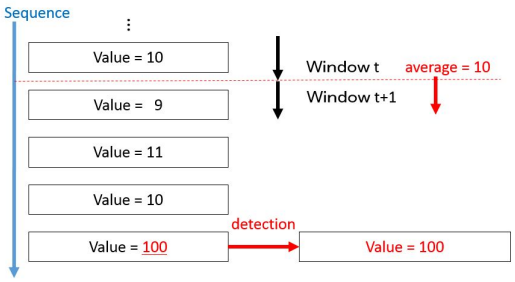

Fig. 3. Window-based change-point detection.

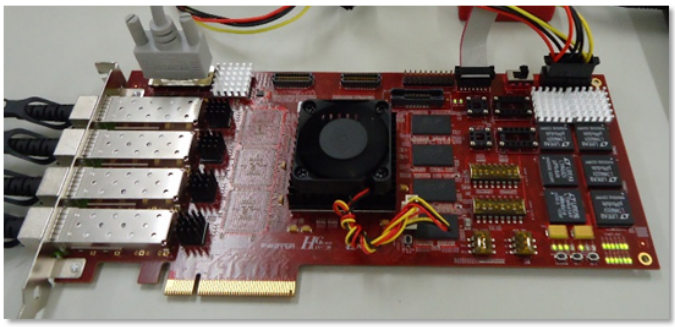

Fig. 4. NetFPGA-10G board.

design for FPGAs was studied in [13]. Scalability issues of the hardware design were discussed.

Electronic commerce is an important application domain for the stream processing. Market data feed arbitration was accelerated by using FPGAs in [14]. The financial transactions require a high speed communication while a single packet loss may cause serious problems. In [14], a dynamic window model was introduced for low-latency and lossless packet processing for reliable market data feed arbitration.

III. Offloading Spark Streaming using FPGA NIC

In this paper, we propose to offload one-at-a-time methodology operations onto FPGA NIC in order to complement the negative aspects of Spark Streaming. By performing stream processing on the FPGA NIC, we can reduce data processing latency compared to the original micro-batch methodology Spark Streaming. By filtering unnecessary data at the NIC, we can reduce the CPU workload (e.g., network protocol processing) and improve the throughput compared to the original Spark Streaming implemented as an application program. We will introduce how our FPGA NIC based approach works with Spark Streaming.

A. Filter Operation

A filter operation discards specific data matched to given patterns. It is useful to reduce data amount and CPU workload by discarding unnecessary data before network processing. Thus, moving the filter operations from an application layer to the FPGA NIC is beneficial.

B. Aggregate Operation in Window

An intra-window aggregate operation groups a series of data in the same window as an input and then it outputs a single value, such as an average value in the window. The output data are transferred to Spark Streaming via a network protocol stack. Typical aggregate operations include average, count, max, min, sum, median, and variance. When the application requires only the aggregated result, the original stream data can be discarded at the NIC; in this case, we can reduce the data amount and CPU workload. Thus, moving the intra-window aggregate operations from an application layer to the FPGA NIC is also beneficial. In Figure 2, values in five packets are aggregated to a single value, resulting in 80% reduction in size.

C. Aggregate Operation between Windows

Some analysis functions require the aggregated or original values of past windows in addition to those in the current window. For example, a change-point detection compares incoming data with the aggregated values of past windows in order to detect change points. Figure 3 illustrates the simple change-point detection using the aggregated values of past windows. More specifically, it uses the average values of past windows as reference data and compares them with an incoming value. If their difference is greater than a certain threshold, the incoming value is detected as a change point. Since Spark Streaming employs a micro-batch methodology, the comparisons between the reference and incoming values are performed at a certain interval which is equal to the length of a micro batch; thus the change-point detection may be delayed depending on the micro-batch length. By performing such a compare operation in one-at-a-time manner with FPGA NIC, we can minimize the detection latency.

Also, the original values of past windows may be required when a specific condition is triggered. For example, when a change point is detected, the original values before and after the change point will be required for the application to analyze the cause. In this case, the past values can be accumulated in DRAMs on the FPGA NIC board and read out when necessary.

IV. Design and Implementation

In this section, we will illustrates design and implementation of one-at-a-time methodology operations on the FPGA NIC incorporated with Spark Streaming.

A. Target FPGA NIC

We use NetFPGA-10G board as a target FPGA NIC. Figure 4 shows the board. It has four 10Gbit Ethernet (10GbE) in-

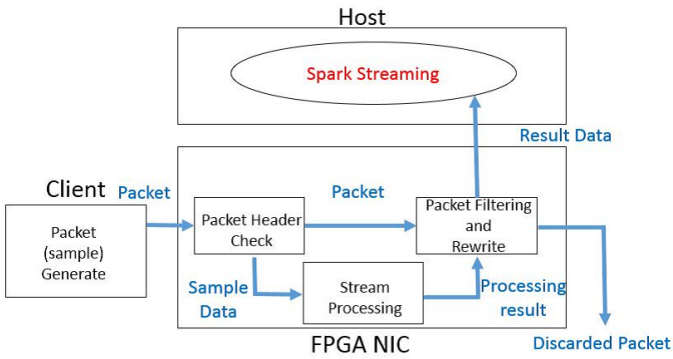

Fig. 5. Stream processing using FPGA NIC.

terfaces. Xilinx 10GbE MAC core is used for these interfaces. Based on Reference NIC design provided by NetFPGA team [5], we implemented one-at-a-time methodology operations on the FPGA NIC. The board has 288MB RLDRAM [1] and we use it to store the aggregated or original values in past windows.

B. Stream Processing Using FPGA NIC

The FPGA NIC provides two major functions: packet processing and stream processing. Figure 5 illustrates the hardware components implemented on the FPGA NIC. As shown in this figure, "Packet Header Check" and "Packet Filter and Rewrite" modules are related to the packet processing, while "Stream Processing" module performs one-at-a-time operations, such as aggregate operation.

1) Packet Processing: In Reference NIC design for NetFPGA-10G, the bit width of AXI (Advanced eXtensible Interface) based internal bus is 256-bit; thus Packet Header Check module receives packets in 256-bit width. When it receives a new packet, it checks the protocol and destination port fields of the packet to see whether it is sent to Spark Streaming. In our implementation, we modified Spark Streaming to support UDP as a transport layer protocol. In this case, UDP packets with a specific port number are detected as Spark Streaming packets. All the other packets, such as ARP (Address Resolution Protocol) packets, are passed to the network protocol stack as usual. Only sample data are extracted from the Spark Streaming packets and passed to Stream Processing module where selected one-at-a-time operations, such as aggregation, are performed. Stream Processing result is passed to Packet Filter and Rewrite module where it is packed as a packet and then transferred to Spark Streaming via a network protocol stack. Thus the one-at-a-time methodology operations are performed at the NIC and the result can be immediately notified to the application.

2) Stream Processing: Here we illustrate two stream processing functions: (1) word count using an intra-window aggregate operation and (2) change-point detection using a inter-window aggregate operation.

(1) Word Count: This function counts the number of each word appeared in a certain time duration. In our experimental system, a server machine has NetFPGA-10G NIC with our packet and stream processing modules and Spark Streaming is running on an application layer. A client machine that has Mellanox 10GbE NIC is connected to the server machine via a SFP+ direct attach copper cable. The client machine sends packets containing words to the server machine. In the stream processing module, a key-value store is implemented where key is corresponding to a word and the value is its counter. When the module receives a word, the value field of the corresponding key is incremented. After a prespecified time duration (e.g., 100usec), a list of (word, count) pairs is transferred to the network protocol stack as a normal Spark Streaming packet. Since Spark Streaming receives the aggregated result as a normal data packet, our FPGA NIC based stream processing is transparent to Spark Streaming. Actually, modifications on Spark Streaming for our proposed system are quite small [2]. After the aggregated result is sent to Spark Streaming, value fields of all the keys are reset to 0. By performing the stream processing at the FPGA NIC, we can reduce the data amount at the NIC and also reduce the CPU workload for the network packet processing and Spark Streaming.

(2) Change-Point Detection: This function monitors incoming data and detects a change point when the received data exceed a certain threshold compared to the past data. We assume one-dimensional numerical data for the input. All the received values are summed up for each window and the average value is calculated at the end of the window by dividing the total value by the number of values in the window. The average values of past windows are used as reference data for the change-point detection. When the stream processing module receives a new value, the value is compared to these reference data; if the difference between them exceeds a certain threshold, it detects a change point. Although this detection method is very simple, we can extend it more sophisticated one. When a change point is detected, the reference data or original data stored in RLDRAM can be also transferred to Spark Streaming for further analysis. By performing the change-point detection in a one-at-a-time manner at the NIC, we can detect the change point immediately when it occurs.

C. Software Processing with Spark Streaming

Spark Streaming provides a receiver function in order to receive stream data. In the original Spark Streaming, a series of data received by the receiver function are stored as a DStream for each micro-batch time interval. In other words, a DStream is a small RDD where stream data during a micro-batch time interval are stored. Then micro-batch operations defined by applications are performed for the DStream.

In this paper, we assume that Spark Streaming deals with a huge amount of sensor data, so UDP is preferred as a transport layer protocol due to the simplicity and low overhead. We modified Spark Streaming to support UDP as a transport layer protocol. More specifically, we introduced a new custom receiver function that can receive UDP packets and transform them into DStream format. Applications first notify the IP address and port number used for Spark Streaming to the custom receiver function so that it can set up the UDP socket.

When stream processing is offloaded to the FPGA NIC, we need to modify applications that use Spark Streaming. For the word count application, for example, the application will

[1] A newer NetFPGA board, called NetFPGA-SUME, has 8GB DRAM.

[2] We only modified Spark Streaming to support UDP as a transport layer protocol.

receive a list of (word, count) pairs since the FPGA NIC performs the word counting. For the change-point detection application, it will receive a change-point value and past reference data only when the change point is detected.

V. Evaluations

A. Evaluation Environment

The proposed system described in the previous sections is implemented on NetFPGA-10G board as an FPGA NIC. Below are the major parameters of the board.

- Xilinx Virtex-5 XC5VTX240TFFG1759-2 (160MHz)
- Four SFP+ interfaces (10Gb x4)
- PCI Express Gen2 x8
- 288MB RLDRAM

Our experiments were performed by using two machines: server and client machines. They are directly connected via a SFP+ direct attach copper cable. Spark Streaming and our applications are running on the server machine. NetFPGA-10G board is mounted in the server machine as a NIC. The server machine has an Intel Core i5-4460 CPU operating at 3.2GHz and 8GB DRAM. As system software, we employ CentOS 6.6, Spark version 1.6.0, Java version 1.7.0_65, and Scala 2.10.5 on the server machine.

In the client machine, a client application program randomly generates sample data and continuously sends them to the server machine as stream data. The server and client applications communicate with each other using UDP as a transport layer protocol since we want to minimize the data transfer overheads. Mellanox 10GbE NIC is mounted in the client machine. The same CPU and DRAM as well as the server are used in the client machine. Ubuntu 15.04 is used as an operating system.

First of all, we implemented the word count and change-point detection applications as described in the previous section on the server and client machines. We confirmed the correct behavior of these applications.

Then we compared the proposed FPGA NIC based approach with the original Spark Streaming application in terms of performance. Regarding the word count application, we measured the maximum throughputs (packets per second) of the proposed FPGA NIC based approach and the original Spark Streaming based approach. To measure the maximum throughput, we employed a 10Gbps hardware packet injector implemented on NetFPGA-10G board. As a client machine, we used the hardware packet injector instead of Mellanox 10GbE NIC for this experiment. For the word count application, the hardware packet injector generates UDP packets that convey randomly-generated words in 10Gbps line rate.

Regarding the change-point detection application, we measured the average latencies (sec) to detect the change points on the proposed FPGA NIC based approach and the original Spark Streaming based approach. The client application continuously sends packets that convey sample data to the server. To generate change points at a specific time interval, value range of the sample data generated by the client is changed. Then we measured the latency to detect the change point at the server side application after the change point was created at the client side application. In this case, the measured change-point detection latency will include a communication latency from the client machine to the server machine. To eliminate this communication latency from the measured change-point detection latency, the client application was executed on the server machine and fed the sample data directly to our stream processing hardware components implemented on the FPGA NIC via PCI Express Gen2 x8. For the original Spark Streaming based counterpart, since Spark Streaming employs micro-batch methodology operations, we set 0.5 sec, 1 sec, and 2 sec as the micro-batch time intervals.

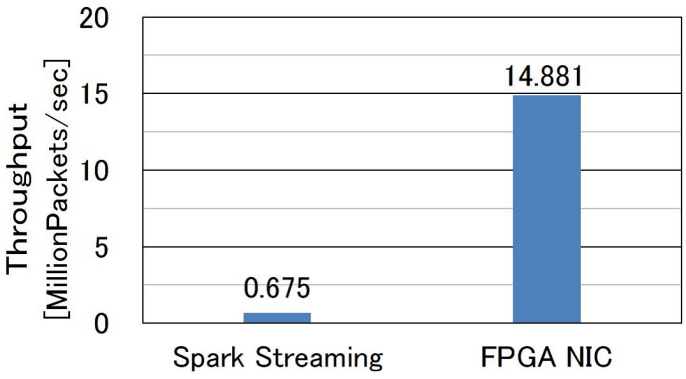

Fig. 6. Maximum throughput for the word count.

B. Area Utilization

All the hardware components including our packet processing and stream processing modules in addition to the original Reference NIC modules were implemented on Xilinx Virtex-5 XC5VTX240TFFG1759-2. The maximum operating frequency satisfies the 160MHz timing constraint. We used Xilinx ISE 13.4 for design synthesis, place, and route of our two implementations: word count and change-point detection.

TABLE II. Area utilization of FPGA NIC.

	Word count	Change-point detection
LUTs	51,045 (34.1%)	50,471 (33.7%)
BRAMs	135 (41.7%)	136 (42.0%)
DSPs	0 (0%)	14 (14.6%)

Table II shows the resource utilizations of LUTs, BRAMs, and DSPs for the two implementations. The word count application does not use any DSPs. As shown in the results, all the resource utilizations are less than 50% of the target FPGA. This means that our proposed systems can be implemented on the FPGA with low hardware overheads and we can employ further sophisticated design options, such as sophisticated change-point detection algorithm as our future work.

C. Throughput for Word Count

Figure 6 shows the maximum throughputs (packets per second) of the word count on the proposed FPGA NIC based approach and the original Spark Streaming based approach. As shown in this graph, the proposed FPGA NIC based approach improves the throughput by 22x compared to the original Spark Streaming. Its throughput is corresponding to 99.8% of the 10Gbps line rate, which means that our implementation achieves the almost optimal performance.

D. Latency for Change-Point Detection

We measured the latencies to detect the change points on the proposed FPGA NIC based approach and the original Spark Streaming based approach. We performed the measurements 100 times for both the cases and calculated the average latencies over the 100 trials. For the original Spark Streaming based approach, since Spark Streaming employs the micro-batch methodology operation, we can estimate the expected latencies depending on the micro-batch lengths (i.e., 0.5 sec, 1 sec, and 2 sec). When the micro-batch length is 1 sec, the expected latency for the original Spark Streaming based approach can be modeled as (0.5 + T_p), where T_p is an actual computation time for the change-point detection by the application software.

TABLE III. LATENCY FOR THE CHANGE-POINT DETECTION (SEC).

	Measured latency	Expected latency
Spark Streaming (0.5 sec)	0.457	0.25 + T_p
Spark Streaming (1 sec)	0.817	0.5 + T_p
Spark Streaming (2 sec)	1.249	1 + T_p
FPGA NIC	0.048	-

Table III shows the measured average latencies and expected latencies for the change-point detection on the four cases. Especially for the original Spark Streaming cases, we can estimate T_p by comparing the measured average latency and the expected latency; as a result, T_p is approximately 0.26sec. On the other hand, the measured average latency for the proposed FPGA NIC based approach is only 0.048sec. This latency is only 5.88% of the original Spark Streaming approach with a 1 sec micro-batch interval. If we assume that the micro-batch interval is zero [3], it is still only 18.46% of the original Spark Streaming approach. These results demonstrate that the proposed FPGA NIC based approach significantly reduces the change-point detection latency. We believe our approach would be beneficial for a wide range of real-time stream analysis applications that require high throughput and/or low latency.

VI. SUMMARY

In this paper, we proposed to offload various one-at-a-time methodology operations onto FPGA NIC and combine it with Spark Streaming in order to complement its negative aspects on the micro-batch methodology processing and application level aggregation/filtering. By performing the stream processing on the FPGA NIC, we can reduce data processing latency compared to the original micro-batch methodology Spark Streaming. By filtering unnecessary data at the NIC, we can reduce the CPU workload (e.g., network protocol processing) and improve the throughput compared to the original Spark Streaming implemented as an application program.

We implemented the word count and simple change-point detection applications on the proposed system. More specifically, we implemented our packet processing and stream processing hardware components on NetFPGA-10G board as an FPGA NIC in conjunction with Spark Streaming applications. The area utilization was modest and the hardware components satisfied the 160MHz timing constraint. We confirmed the correct behaviors of these applications on the real machines. Experimental results using real machines demonstrated that the word count throughput was improved by 22x compared to the original Spark Streaming. This throughput is 99.8% of the 10Gbps line rate, which means that our implementation achieves the almost optimal performance. In addition, the change-point detection latency was reduced to only 5.88% of the original Spark Streaming approach with a 1 sec micro-batch interval. These results demonstrated that the proposed FPGA NIC based approach significantly improved the performance of these applications and it would be beneficial for a wide range of real-time stream analysis applications that require high throughput and/or low latency.

To the best of our knowledge, this paper is the first work to offload Spark Streaming on FPGA NIC. As the aim of this paper is to demonstrate the performance improvement by the hardware offloading, we implemented relatively simple applications on the FPGA NIC. We are now planning to implement more sophisticated stream analysis operations on the FPGA NIC as a future work.

Acknowledgements This work was supported by JST PRESTO and JSPS KAKENHI Grant Number JP16H02793.

[3] This is an extreme assumption.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*, Jun. 2008.

[2] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'13)*, Nov. 2013, pp. 423–438.

[3] Y. Xua, E. Frachtenbergb, and S. Jiang, "Building a High-Performance Key-Vluec Cache as an Energy-Efficient Apppliance," *Performance Evaluation*, vol. 79, pp. 24–37, Sep. 2014.

[4] Y. Tokusashi and H. Matsutani, "A Multilevel NOSQL Cache Design Combining In-NIC and In-Kernel Caches," in *IEEE International Symposium on High Performance Interconnects (Hot Interconnects 24)*, Aug. 2016, pp. 60–67.

[5] "The NetFPGA Project," http://netfpga.org/.

[6] Nathan Marz and James Warren, *Big Data: Principles and Best Practices of Scalable Real-Time Data Systems*. Manning Publications, 2015.

[7] "Apache Storm," http://storm.apache.org/.

[8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, Apr. 2012, pp. 15–28.

[9] A. Hayashi, Y. Tokusashi, and H. Matsutani, "A Line Rate Outlier Filtering FPGA NIC using 10GbE Interface," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 4, pp. 22–27, Sep. 2015.

[10] P. Benacek, R. B. Blazek, T. Cejka, and H. Kubatova, "Change-Point Detection Method on 100 Gb/s Ethernet Interface," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'14)*, Jun. 2014, pp. 245–246.

[11] R. Mueller, J. Teubner, and G. Alonso, "Streams on Wires: A Query Compiler for FPGAs," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'09)*, Aug. 2009, pp. 229–240.

[12] Y. Oge, T. Miyoshi, H. Kawashima, and T. Yoshinaga, "Design and Implementation of a Handshake join Architecture on FPGA," *IEICE Transactions on Information and Systems*, vol. E95-D, no. 12, pp. 2919–2927, Dec. 2012.

[13] Y. Oge, M. Yoshimi, T. Miyoshi, H. Kawashima, H. Irie, and T. Yoshinaga, "An Efficient and Scalable Implementation of Sliding-Window Aggregate Operator on FPGA," in *Proceedings of the International Symposium on Computing and Networking (CANDAR'13)*, Dec. 2013, pp. 112–121.

[14] S. Denholm, H. Inoue, T. Takenaka, T. Becker, and W. Luk, "Low Latency FPGA Acceleration of Market Data Feed Arbitration," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP'14)*, Jun. 2014, pp. 36–40.