Optimizing Hardware-Based Privacy-Preserving MapReduce

Han-Yee Kim, Rohyoung Myung, Sangwoo Park, Sukyong Choi, Heonchang Yu, Taeweon Suh
Department of Computer Science and Engineering,
Korea University
145 Anam-ro, Seongbuk-gu, Seoul, Korea
{hanyeemy, mry1811, psw0113, csukyong, yuhc, suhtw}@korea.ac.kr

Jungha Lee
Disaster Information Service Lab
Korea Institute of Science and Technology Information (KISTI)
245 Daehak-ro, Yuseong-gu, Daejeon, Korea
jungha07@kisti.re.kr

1 ABSTRACT

For big data processing, it is reasonable to offload complicated jobs to cloud for efficiency. However, due to the security attacks from rogues, there is a risk of sensitive information leakage [on cloud system side. Especially, if the data set includes privacy sensitive information, users may hesitate to utilize the cloud services due to the security concerns. In this paper, we introduce a hardware-based privacy preserving MapReduce framework[1] and enhance it with two key techniques: Separate barrier and module combining scheme. Both schemes not only enhance the system performance by efficient data flow control but optimize the hardware resource utilization to fit in the capacity of hardware as well. The implementation of system is executed on Zynq-7000 programmable SoC device[2] from Xilinx. The combining scheme boasts of at least a 2.20x faster performance in the experiment.

2 PROPOSED SYSTEM AND TECHNIQUES

2.1 Proposed Hardware-Based Privacy-reserving MapReduce System

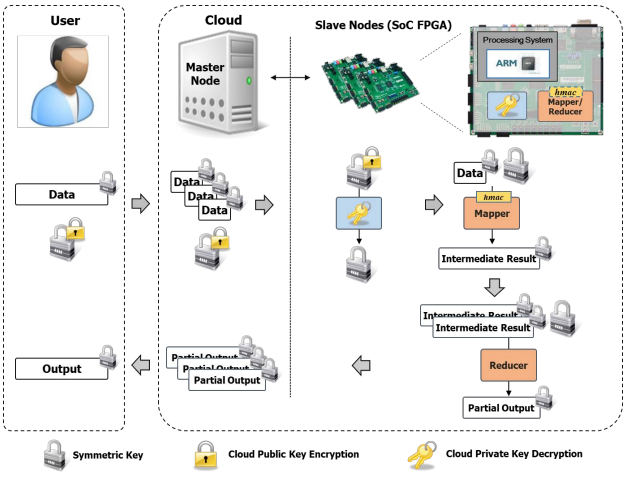


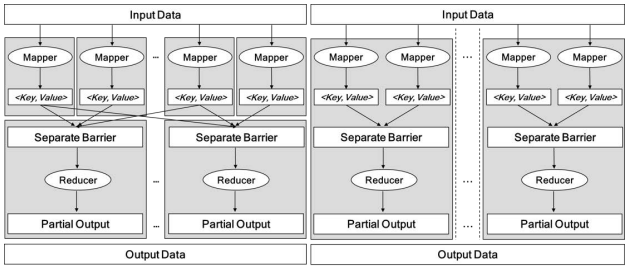
Figure 1: Overview of Secure Hardware MapReduce System

As shown in Fig .1, the proposed system is composed of an x86 server machine as a master node, and programmable SoCs as slaves.

In our scenario, with CAD tools provided by FPGA vendor, user can automatically create hardware mappers and reducers with AES crypto modules via High Level Synthesis (HLS) tool[3]. They are integrated to a single bitstream file and the file is re-encrypted with AES key. User also should encrypt his data with AES. After that, the AES key is further re-encrypted with RSA[4] public key of FPGA. Then, all the encrypted data, encrypted hardware design, and re-encrypted AES key are transferred to the cloud. In the system, the root of trust is the re-encrypted AES key which will be decrypted inside FPGA by its private key. Because system configuration as well as data processing is done as enclaved format in tamper-resistant FPGA, the proposed system is secure from rogue users even they get the root authority of system.

2.2 Optimizing Schemes on the Hardware-based Privacy-Preserving MapReduce

The separate barrier and combining scheme are shwon in Fig .2.



(a) Separate Barrier Scheme (b) Combining Scheme

Figure 2: Proposed techniques on Secure MapReduce Framework

- Separate barrier: It eases the bottleneck of global barrier between mapper and reducer by localizing them. The separate barrier is implemented with a simple counter in the head of each reducer, where the counter is initially set to the number of inputs. If data set from a mapper comes in, the counter is decreased by one. When all the inputs of reducer are available, the counter reaches 0 and reducer starts the task processing. After finishing reducer job, separate barrier counter will be re-initialized to the number of inputs.

- Module combining is conditional scheme. If there is no complicated shuffling between mappers and reducers so that they can be serialized by certain input data chunk, the mappers and reducers can be combined and configured on a single FPGA board with separate barrier.

3 IMPLEMENTATION AND EXPERIMENT RESULT

3.1 Target Applications and its Parameters

In the experiment, we use two target applications (K-means and DNA sequencing) for implementation. K-means clustering is one of the most commonly used algorithms in big data processing. Given a set of *n* data where each data corresponds to a point in a *d*-dimensional space, its goal is to cluster closest data points, aggregating data set into *k*-groups. DNA sequencing is an application where users' DNAs are compared against reference DNAs. After data processing, the application scores the difference between them. DNA sequence is divided into a certain size of *block*s. In the mapper phase, each block is further divided by certain *string* size by enumerating all possible consecutive subsequences. The parameters of the target applications are shown in Table 1.

Table 1: Parameters in Target Applications

Applications	Parameters	
K-means	# Centroid (16, 32, 48)	# Dimension (4, 6, 8)
DNA sequencing	Block size (16, 32, 48)	String size (6, 8 , 10)

3.2 Optimizing Method of FPGA via HLS

We implemented our proposed schemes using Zynq-7000 [2] (xc7z020clg484-1) programmable SoC device. Zynq-7000 has two interacting parts: processing system (dual Cortex-A9s) and Programmable Logic (PL). The target applications are originally written in C. To convert the C code to hardware, we use a CAD tool called Vivado High Level Synthesis (HLS) from Xilinx. The *hmac[5]* and RSA[4] decryption module are also originally written in C and converted to Verilog via Vivado HLS in the same way. Note that HLS cannot convert dynamic functions to hardware like *malloc()*. In addition, some libraries are also not supported. Therefore, our target applications are partially tuned to adjust the hardware conversion mechanism on Zynq-7000.

For the performance tuning, we insert some directives in the original C code. The *PIPELINE* directive allows efficient data processing by increasing hardware utilization. The *ARRAY_PARTITION* and *RESOURCE* directives allow the Cortex-A9 to access IP data as memory mapped registers. The input of RSA decryption module, the input of AES decryption module, and the output of AES encryption module are configured with those directives for access from Cortex-A9.

In the experiment, to maximize the throughput of mapper and reducer, AES crypto modules are configured as many as possible in each programmable SoC. However, zynq has limited resources. If the implemented hardware module exceeds the capacity of it, the module cannot be configured as maximum throughput. Algorithms for handling this situation is shown in Algorithm 1 (for separate barrier) and Algorithm 2 (for combining scheme).

ALGORITHM 1. Integrated MapReduce Module Configuration Algorithm for Separate Barrier Scheme

D = # bits of AES decrypt module
E = # bits of AES encrypt module
$i = 1$;
$j = 1$;
$d_i = \lfloor D/(128 * i) \rfloor$;
$e_j = \lfloor E/(128 * j \rfloor$;
repeat
if (d_i**cost*(AES decrypt) > e_j**cost*(AES encrypt))
  $i = i+1$;
else
  $j = j+1$;
until (d_i**cost*(AES decrypt) + *cost*(mapper or reducer) + e_j**cost*(AES encrypt) < capacity)
configure SoC;

ALGORITHM 2. Integrated Combining Module Configuration Algorithm

M = # of mappers to one reducer
D = # bits of AES decrypt module
E = # bits of AES encrypt module
T_m = # of cycle for mapper
T_d = # of cycle of AES decrypt module
T_e = # of cycle of AES encrypt module
$i = 1$;
$j = 1$;
$d_i = \lfloor D/(128 * i) \rfloor$;
$e_j = \lfloor E/(128 * j \rfloor$;
$m_i = \lfloor M/i \rfloor$;

repeat
if (d_i **cost*(AES decrypt) + m_i **cost*(mapper) + *cost*(reducer) + e_{j+1}**cost*(AES encrypt) < capacity)
  $j = j+1$;
else if (d_{i+1}**cost*(AES decrypt) + m_{i+1}**cost*(mapper) + *cost*(reducer) + e_j**cost*(AES encrypt) < capacity)
  $i = i+1$;
else
  if $(\{(d_i - d_{i+1}) * cost(\text{AES decrypt}) + (m_i - m_{i+1}) * \text{cost(mapper)}\} / (T_m + T_d) < (e_j - e_{j+1}) * cost(\text{AES encrypt}) / T_e)$
    $j = j+1$;
  else
    $i = i+1$;
until (d_i **cost*(AES decrypt) + m_i **cost*(mapper) + *cost*(reducer) + e_j**cost*(AES encrypt) < capacity)
configure SoC;

3.3 Evaluation and Analysis

The integrated module configurations applying Algorithm 1 and Algorithm 2 are shown in Table 2 and Table 3, respectively.

Table 2: Integrated Module Configuration Applying Algorithm 1

Applications	Modules	Parameters	FPGA Configuration
K-means	Mapper	4 Dimension	Fully Configured
		6 Dimension	Fully Configured
		8 Dimension	Fully Configured
	Reducer	16 Centroid	Fully Configured
		32 Centroid	AES Decrypt 1/2
		48 Centroid	AES Decrypt 1/2
DNA sequencing	Mapper	16 Block, 6 String	Fully Configured
		16 Block, 8 String	Fully Configured
		16 Block, 10 String	Fully Configured
		32 Block, 6 String	AES Encrypt 1/2
		32 Block, 8 String	AES Encrypt 1/2
		32 Block, 10 String	AES Encrypt 1/2
		48 Block, 6 String	AES Encrypt 1/2
		48 Block, 8 String	AES Encrypt 1/2
		48 Block, 10 String	AES Encrypt 1/2
	Reducer	16 Block, 6 String	AES Decrypt 1/2
		16 Block, 8 String	Fully Configured
		16 Block, 10 String	Fully Configured
		32 Block, 6 String	AES Decrypt 1/3
		32 Block, 8 String	AES Decrypt 1/3
		32 Block, 10 String	AES Decrypt 1/3
		48 Block, 6 String	AES Decrypt 1/4
		48 Block, 8 String	AES Decrypt 1/5
		48 Block, 10 String	AES Decrypt 1/5

Table 3: Integrated Module Configuration Applying Algorithm 2

Applications	Parameters	FPGA Configuration
K-means	4 Dimension, 16 Centroid	AES Decrypt, Map 1/3
	4 Dimension, 32 Centroid	AES Decrypt, Map 1/5
	4 Dimension, 48 Centroid	AES Decrypt, Map 1/7
	6 Dimension, 16 Centroid	AES Decrypt, Map 1/3
	6 Dimension, 32 Centroid	AES Decrypt, Map 1/6
	6 Dimension, 48 Centroid	AES Decrypt, Map 1/9
	8 Dimension, 16 Centroid	AES Decrypt, Map 1/4
	8 Dimension, 32 Centroid	AES Decrypt, Map 1/8
	8 Dimension, 48 Centroid	AES Decrypt, Map 1/12
DNA sequencing	16 Block size, 6 Sub	Fully Configured
	16 Block size, 8 Sub	Fully Configured
	16 Block size, 10 Sub	Fully Configured
	32 Block size, 6 Sub	Fully Configured
	32 Block size, 8 Sub	Fully Configured
	32 Block size, 10 Sub	Fully Configured
	48 Block size, 6 Sub	Fully Configured
	48 Block size, 8 Sub	Fully Configured
	48 Block size, 10 Sub	Fully Configured

Fig .3 shows the hardware resource utilization of integrated mapper and reducer modules when applying Algorithm 1. In most cases, it is insufficient BRAM resources that cause a modification of the initial configuration. As the AES module occupies the majority of BRAM resources, it is split by Algorithm 1. The execution time of integrated mapper and reducer module for separate barrier scheme is shown in Fig .4 and it implies that *hmac* is the critical path of both target applications in all the cases of mapper. Because reducers do not contain *hmac* module, all the reducers are relatively faster than the corresponding mapper.

Fig .5 shows the hardware cost of integrated combined module with the configurations of Algorithm 2. Fig .6 shows the execution time of integrated combined module when applying combining scheme. Even not considering the network traffic between programmable SoC and Cortex-A9, the combining scheme shows quite remarkable speedup. Especially in DNA Sequencing, as assuming the number of utilizing boards is same, the highest speedup (293.08x) is achieved with the case of *16 Block size and 10 String size* in DNA Sequencing whereas the lowest speedup (2.20x) is achieved with the case of *8 dimension and 48 centroids* in K-means. The elimination of AES crypto module makes more MapReduce modules can be accommodated and the elimination of *hmac* module reduces the processing time of mappers dramatically.

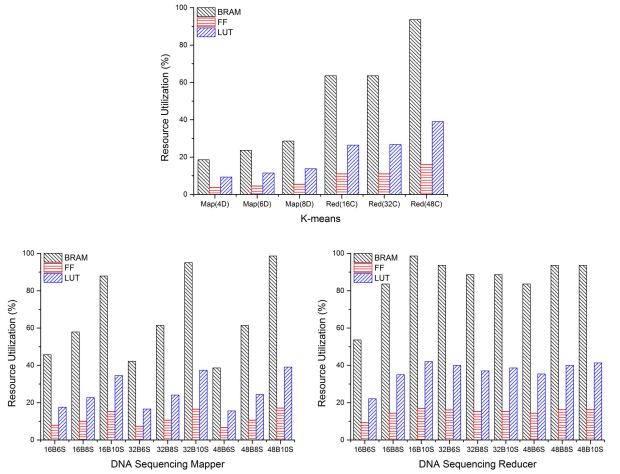


Figure 3: Hardware resource utilization of integrated modules applying separate barrier

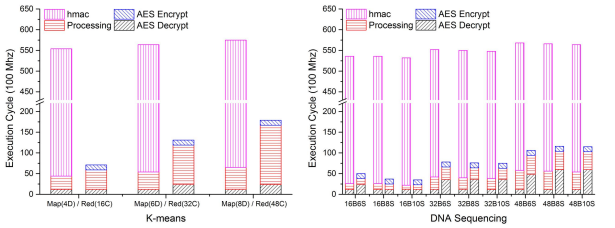


Figure 4: Execution time of integrated modules applying separate barrier

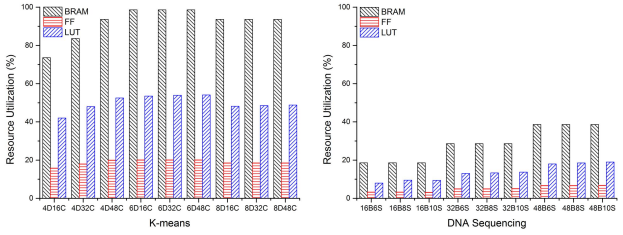


Figure 5: Hardware resource utilization of integrated modules applying combining scheme

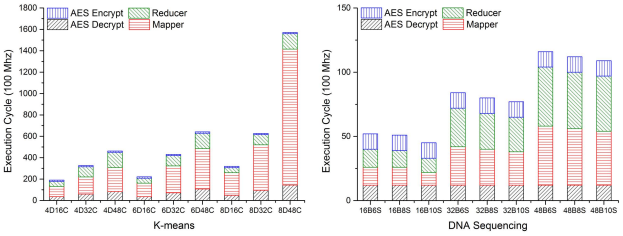


Figure 6: Execution time of integrated modules applying combining scheme

4 RELATED WORK

There have been efforts of accelerating MapReduce in two ways: the first takes advantage of off-the-shelf fixed parallel computing resources such as GPU, Xeon Phi, and many-core processors[6-10] and the second one utilizing reconfigurable hardware[11-13]. In the reconfigurable camp, Mershad et al[11] proposed a new service model where users can choose to pay a premium for faster data processing by exploiting FPGAs. Lin et al[12] proposed an eight Zynq-based Hadoop cluster, referred to as ZCluster. Shan et al[13] implemented a MapReduce framework on FPGA, referred to FPMR. In FPMR, the whole process of the mapper and reducer is conducted on the FPGA side, and the mapper and reducer are scheduled by hardware queue. There are a few studies on reducing the synchronization overhead of MapReduce by eliminating the global barrier[14, 15]. Elteir et al [14] proposed a hierarchical reduction, where the map and reduce processing is overlapped at the inter-task level and the reduce task gets started as soon as a certain number of map tasks complete. The partial outputs from reducers are aggregated following a tree hierarchy. Verma et al [15] classified reduce operations according to applications' characteristics, and eliminated the global barrier in Hadoop by using tree based scheme.

ACKNOWLEGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2017M3C4A7081956).

REFERENCES

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM. 2008;51(1):107-13.
[2] Crockett LH, Elliot RA, Enderwitz MA, Stewart RW. The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc: Strathclyde Academic Media; 2014.
[3] Feist T. Vivado design suite. White Paper. 2012;5.
[4] Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM.
[5] Krawczyk H, Canetti R, Bellare M. HMAC: Keyed-hashing for message authentication. 1997.
[6] Chen SY, Lai CF, Hwang RH, Chao HC, Huang YM, editors. A multimedia parallel processing approach on GPU MapReduce framework. Ubi-Media Computing and Workshops (UMEDIA), 2014 7th International Conference on; 2014: IEEE.
[7] Fang W, He B, Luo Q, Govindaraju NK. Mars: Accelerating mapreduce with graphics processors. IEEE Transactions on Parallel and Distributed Systems. 2011;22(4):608-20.
[8] Lu M, Zhang L, Huynh HP, Ong Z, Liang Y, He B, et al., editors. Optimizing the mapreduce framework on intel xeon phi coprocessor. Big Data, 2013 IEEE International Conference on; 2013: IEEE.
[9] Teodoro G, Kurc T, Kong J, Cooper L, Saltz J, editors. Comparative performance analysis of Intel (R) Xeon Phi (TM), GPU, and CPU: a case study from microscopy image analysis. Parallel and Distributed Processing Symposium, 2014 IEEE 28th International; 2014: IEEE.
[10] Honjo T, Oikawa K, editors. Hardware acceleration of hadoop mapreduce. Big Data, 2013 IEEE International Conference on; 2013: IEEE.
[11] Mershad K, Kaitoua AR, Artail H, Saghir MA, Hajj H, editors. A framework for multi-cloud cooperation with hardware reconfiguration support. Services (SERVICES), 203 IEEE Ninth World Congress on; 2013: IEEE.
[12] Lin Z, Chow P, editors. Zcluster: A zynq-based hadoop cluster. Field-Programmable Technology (FPT), 2013 International Conference on; 2013: IEEE.
[13] Shan Y, Wang B, Yan J, Wang Y, Xu N, Yang H, editors. FPMR: MapReduce framework on FPGA. Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays; 2010: ACM.
[14] Elteir M, Lin H, Feng W-c, editors. Enhancing mapreduce via asynchronous data processing. Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on; 2010: IEEE.
[15] Verma A, Cho B, Zea N, Gupta I, Campbell RH. Breaking the MapReduce stage barrier. Cluster computing. 2013;16(1):191-206.