HeteroYARN: A Heterogeneous FPGA-Accelerated Architecture Based on YARN

Ruixuan Li, *Member, IEEE*, Qi Yang, Yuhua Li, Xiwu Gu, Weijun Xiao, and Keqin Li, *Fellow, IEEE*

Abstract—In recent years, the heterogeneous distributed platform integrating with FPGAs to accelerate computation tasks has been widely studied to deal with the deluge of data. However, most of current works suffer from poor universality and low resource utilization that run specific algorithms with the highly customized structure. Moreover, there are still many challenges, such as data curation, task scheduling, and resource management, which further limit the scalability of a CPU-FPGA distributed platform. In this paper, we present HeteroYARN, an FPGA-accelerated heterogeneous architecture based on YARN platform, which provides resource management and programming support for computing-intensive applications using FPGAs. In particular, the HeteroYARN abstracts FPGA accelerators as general resources and provides programming APIs to utilize those accelerators easily. Our HeteroYARN simplifies the request and usage of FPGA resources to enhance the efficiency of the heterogeneous framework while maintaining previous workflow unchanged. Experimental results using two representative algorithms, K-means and Naive Bayes classifier, which are accelerated by FPGAs, demonstrate the usability of the HeteroYARN framework and show performance speedup improvement by 7.5x (K-means) and 2.3x (Naive Bayes) respectively compared to conventional CPU-only applications provided by Mahout.

Index Terms—Heterogeneous system, heterogeneous FPGA architecture, FPGA-accelerated computing, data-intensive computing, YARN

1 Introduction

The massive data processing demand in the big data era is accompanied by the increasing amount of training data and computation complexity, which has brought enormous pressure to the traditional CPU-based data processing platform [1]. With the emergence of "dark silicon" [2], limitations that prevent powering up all processors are encountered, even when using multi-core design. For effective analysis and interpretation of the big data, scalable machine learning methods are required to overcome the space and time bottlenecks [3]. As the CPU-only architecture can no longer meet the growing computational performance requirements, researchers are seeking better solutions to satisfy the continually increasing computation demands. Thus, the rapid improvements in processing that we have expected in the Moore's law era must now come through innovations in computer architecture. One path left to continue the improvements in performance of processors is developing domain-specific processors [4]. With the excellent characteristics of low power, high efficiency, and reprogrammability to customize high-performance computing, integrating field-programmable gate array (FPGA) accelerators into computing systems has become a better choice among various solutions. Examples of FPGAs in data centers, such as Microsoft's Catapult [5] and Intel's HARP [6], have already emerged, improving the computing capability of the cluster with less energy consumption. Integrating FPGAs into a distributed cluster is considered as one of the most promising approaches to sustain the computation demand growth.

In contrast to the serial instruction executions used by the general-purpose processor, internal computation components enable FPGAs to achieve device-level parallelism. As a semi-custom circuit chip in the field of Application Specific Integrated Circuit (ASIC), its on-chip logic gates are rich in resources, and the way of connection can be changed by programming. FPGAs can be customized to accelerate various applications' computation-intensive tasks according to their reprogramming ability. These features make it a formidable component for datacenters [7], achieving higher computational performance at lower clock speeds and power consumption. Hence, it is feasible to enhance the computation power of the distributed cluster while reducing power consumption, through integrating FPGA accelerators into a cluster to deal with computationally intensive tasks [5].

Many kinds of parallel computing platforms have been proposed, including graphics processing unit (GPU) clusters [8], FPGA clusters [9] and GPGPU/FPGA [10] mixed clusters, to improve the performance of data processing for different aspects of computing tasks. However, there are still several key issues needed to be solved along with the emerging trend of FPGA-enabled computing cluster.

- *High development difficulty*. The structure of a heterogeneous cluster is not transparent to developers, and

- *R. Li, Q. Yang, Y. Li, and X. Gu are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: {rxli, ayang7, idcliyuhua, guxiwu}@hust.edu.cn.*
- *W. Xiao is with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284. E-mail: wxiao@vcu.edu.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 12 Sept. 2018; revised 23 Feb. 2019; accepted 25 Feb. 2019. Date of publication 15 Mar. 2019; date of current version 27 July 2020. (Corresponding author: Yuhua Li.)
Recommended for acceptance by T. Kosar.
Digital Object Identifier no. 10.1109/TPDS.2019.2905201

the development cost with FPGAs is substantial that advanced hardware and Register Transfer Language (RTL) programming knowledge are required, and lead to long development cycles and high cost.

- *Low utilization.* Current heterogeneous architectures are highly customized for specific algorithms, which undermine the homogeneity and reduce overall flexibility with low resource utilization. Hadoop1.0 based framework shield underlying details to reduce development costs. Nevertheless, a whole cluster can only employ highly customized FPGAs, limited by the homogeneous cluster assumptions.
- *Poor generality and flexibility.* Because of the constrains of the platform in resource management and job scheduling coupling, Hadoop based research is mostly applied to accelerate the implementation of MapReduce framework algorithm, and its cluster architecture cannot support other popular computing frameworks.

The FPGAs enable the implementation of custom computing tasks by standard software languages, but a specialized large scale deployment without flexibility would not be a good choice [11]. Hence, it is needed to propose a generic approach that provides easy access to FPGA resources for developers, reducing programming difficulties while managing the heterogeneous cluster transparently with good scalability and flexibility.

To address these issues, we create a heterogeneous CPU-FPGA distributed platform, which leverages the capabilities of the Yet Another Resource Negotiator (YARN) framework [12], and combine it with the power of FPGAs to achieve a high-performance, low-power and efficient computing platform for accelerating computation-intensive algorithms, namely HeteroYARN. Based on the YARN framework, we design new resource management strategies and scheduling mechanisms to suit for the heterogeneous CPU-FPGA cluster by modifying the related modules, including ResourceManager (RM) and NodeManager (NM), and provide development interfaces for easy usage of FPGAs as well. Furthermore, we deploy a heterogeneous cluster using Xilinx FPGA boards to demonstrate the validity and practicability of our HeteroYARN framework, which provides the resource abstraction and programming support for easy and efficient FPGA usage. Our goal in building HeteroYARN is to find a better way to manage FPGA resources in heterogeneous environments, and try to achieve significant performance improvements in computing-intensive algorithms.

However, FPGA is specialized as a special computing acceleration hardware, which is quite different from the way CPU and GPU work. CPU and GPU have instruction sets that can load different sequences to execute different programs, while excessive programming efforts are needed to reprogram an FPGA into a different functional accelerator to execute different programs. Moreover, it will take several hours to compile an FPGA program. Therefore, it would be better to pre-equip FPGA accelerators in advance to diminish the overhead. From the view of cluster resource management, these FPGAs with different algorithms cannot be regarded as one kind of computing resources, even though they have the same hardware structure. A generic system to support various applications must deal with all kinds of resources, including diverse FPGAs programmed for different tasks. In order to ensure resource utilization, resource management platform should have the ability to recognize and schedule different kinds of accelerator resources, which will bring huge FPGA deployment and maintenance burden. Therefore, in this paper, we propose a unified FPGA communication interface and device virtualization functions to coordinate the FPGA communication and multi-process FPGA accelerator sharing, greatly enhancing the efficiency of application development.

To this end, we build up an integrated platform for the accelerated applications using FPGAs to shorten the computing time, by extending resource manager and task scheduler of the YARN platform. The heterogeneous cluster hardware architecture details are hidden to the users, yielding a significant improvement in programmability. Besides, the algorithm implementations on HeteroYARN can take the advantage of FPGA accelerator resources by modifying the computing dataflow that offloads computation-intensive tasks to FPGAs. As an application scenario, we can adopt the CPU-FPGA heterogeneous cluster architecture to accelerate the machine learning algorithms.

In summary, the paper presents a heterogeneous CPU-FPGA system on YARN and makes the following contributions.

1) We propose a heterogeneous cluster resource management framework called HeteroYARN, including a new resource representation scheme that manages logical FPGA accelerator functionality for better scheduling decision making, the profile interfaces for users to adjust resource capacity easily in computing nodes, and the HeteroScheduler algorithm for global multi-kind resource management to alleviate the scheduling overhead. Also, we propose a two-way link to communicate and transfer data between the CPU and FPGAs in the heterogeneous cluster.
2) An implementation of MapReduce framework is designed based on HeteroYARN to enable computation-intensive applications to perform data processing tasks on FPGA accelerators and maintain the original computing model unchanged, which is capable of combing MapReduce workflow and FPGA computation power, and provides necessary programming APIs to support the usage of FPGA accelerators.
3) We implement some computing-intensive algorithms on the HeteroYARN framework. We modify the data processing flow, offloading the computationally intensive tasks to the FPGA accelerators, to achieve the purpose of improving performance while avoiding rising the development costs.
4) We evaluate our architecture by the experimental results of K-means and Naive Bayes implementations, and verify the effectiveness of resource management and multi-job scheduling with more supplementary experiments. We further analyze the performance bottlenecks of our FPGA-accelerated environments based on the experimental results.

The rest of the paper is organized as follows. Section 2 introduces the background knowledge and the related works. Section 3 presents the overall system design of the HeteroYARN. Section 4 provides a detailed description about the heterogeneous MapReduce implementations on

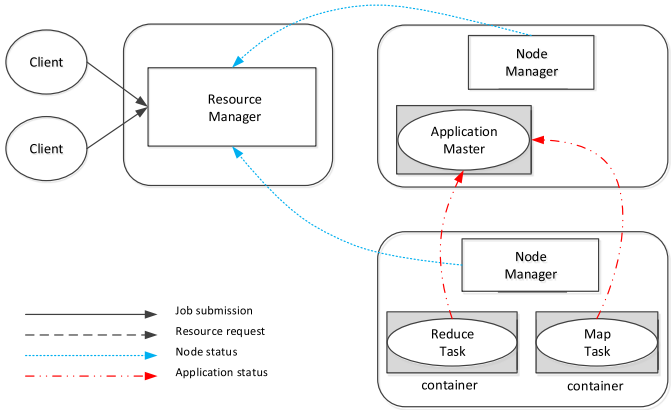


Fig. 1. Example of submitting MapReduce jobs to the resource manager of YARN platform.

HeteroYARN platform. Section 5 shows the experimental results obtained and evaluates the performance of the system. Finally, Section 6 concludes this paper and highlights our future works.

2 Related Work

Considering the excellent performance of FPGA chips on improving computing performance, the academic community has done a substantial amount of previous studies In this section, we first introduce the Hadoop YARN, which is the architecture we extend for management and utilization of FPGA resources in the heterogeneous environment. Then, we describe the characteristics and advantages of FPGA accelerators, existing technical obstacles and existing heterogeneous systems especially designed for FPGAs.

2.1 Distributed Resource Management on YARN

Yet Another Resource Negotiator [12] is a widely used cluster resource management layer in the Hadoop system that allocates resources, such as CPU and memory, to multiple big data applications (or jobs). Compared to the previous Hadoop version, YARN decouples the cluster resource management and task scheduling to achieve the cluster sharing, scalability and reliability. The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons, as shown in Fig. 1, central RM and several per-node NMs. Each NM typically manages resources available on a single node, and periodically reports to the RM, which arbitrates all the available cluster resources, and thus helps manage the distributed applications running on the YARN system. Periodic reports also serve as the basis for monitoring the health of the entire cluster so that RM can notify related applications when the failure occurs. From Fig. 1, we can see the ApplicationMaster (AM) instance, which is responsible for negotiating resources from RM and working with the NMs to start the allocated containers, and executing the MapReduce jobs on YARN architecture.

Considering the resource utilization, maintenance cost and data sharing, developers tend to deploy multiple computing framework to a common cluster, and different resource requests (CPU, memory, I/O and so on) make it hard to avoid interfering with each other. In order to reduce the inefficiencies caused by request competition, YARN is positioned as a data operating system, providing better resource management and scheduling for all types of applications in different framework such as MapReduce [13] and Spark [14]. YARN evolved from MapReduce, which plays an important role in big data processing, but cannot be regarded as a universal resource management system now. Meanwhile, there are several resource managers that are similar to YARN such as Mesos [15], Omega [16] and Borg [17]. However, none of them can directly support the management of FPGA accelerator resources.

To manage heterogeneous computing resources for different workloads and organizations, YARN recently introduces a mechanism called label-based scheduling [18]. Label-based scheduling provides job placement control on a multi-tenant hadoop cluster [19]. An administrator can specify labels for node and then composes task queue or job labels based on the node labels, to control exactly which nodes are chosen to run jobs with specified label. We have adopted a similar resource management strategy, but have a completely independent design and implementation. Specifically, both scheduling mechanisms consider to mitigate the scheduling overhead. The HeteroScheduler algorithm is mainly based on the resource dimension extension and accelerator resource annotation. Hence, finer-grained scheduling decisions can be made to optimize the system throughput in global. In other words, the HeteroScheduler can be regarded as the extension and implementation of label-based scheduling for the heterogeneous FPGA cluster. More details are given in Section 3.2.

Furthermore, there are already many works focusing on integrating GPUs at cluster scale, mainly in combination of GPU cluster and different existing computing platforms, such as Hadoop [20], [21], [22], [23], Spark [24], Flink [25]. However, compared to GPUs, there is no instruction system in FPGAs and only few ready-made libraries available. Therefore, these GPU management techniques cannot be used to manage FPGA systems directly. Hence, we design the message format and a interaction system carefully to improve the efficiency of the entire system in our work.

2.2 Distributed FPGA Cluster System

A FPGA accelerator is a reconfigurable integrated circuit with much lower power consumption compared to CPUs and GPUs. Since an FPGA is essentially customizable hardware, it can achieve significant performance speedup at low clock frequency. Owing to FPGAs' energy efficiency, it has been widely adopted for accelerating the computation intensive kernels in standalone applications [26], [27]. Most FPGA-based studies leverage FPGAs to implement specific applications [28] by integrating FPGAs into the data processing and taking the accelerators as the co-processor to achieve promising performance improvement. Applications related to data center also gradually focus on the heterogeneous architectures [29], which is similar to the framework for creating network FPGA clusters in a heterogeneous cloud data center [30], and a hybrid CPU-FPGA databases for accelerating relational engine [31].

An FPGA implementation design is usually based on hardware description languages (HDLs), such as Verilog and VHDL, and requires a comprehensive knowledge about hardware. High Level Synthesis (HLS) techniques have been

developed aiming at (semi-)automatically generating hardware implementations of specifications written in high level languages [32]. However, the learning-curve for FPGA programming is still very steep for software programmers, since optimizing implementation requires a significant amount of FPGA-specific knowledge.

Some studies [6], [33] assume that all nodes in the cluster are equipped with the same functional FPGA accelerators, so that the computing logic for accelerator can be embedded in the subtasks during the programming phase. They do not consider the heterogeneous resource management, and ignore the task scheduling. FPMR [34] and Melia [35] are examples of the remarkable works. FPMR [34] develops a MapReduce framework on FPGA to eliminate the data communication. Although FPMR provides programming abstraction, hardware architecture, and basic building blocks to developers, users still have to write customized map/reduce functions. Melia [35] utilizes the OpenCL programming framework, the general-purpose accelerator language, to abstract FPGAs behind the MapReduce interfaces in C. However, the performance evaluation of multi-node clusters only depends on simulation results instead of practical implementations.

Besides heterogeneous MapReduce framework, some studies have integrated Peripheral Component Interconnect Express (PCIE) based high-performance FPGA accelerators into Spark running on commodity clusters. SparkCL [36] generates an OpenCL kernel for Altera FPGAs automatically and executes FPGA accelerators on Spark. However, the programming model of SparkCL discloses low-level OpenCL APIs, such as thread ID, to users and only supports primitive data types. Chen et al. [37] take genome sequence alignment as case study to deploy FPGA accelerators onto Spark. Blaze [38] abstracts FPGA accelerator as a service (FaaS), and provides a set of clean programming APIs for applications to access the FPGA accelerators easily that run on top of Apache Spark and Hadoop YARN. In comparison, our architecture can get better computational efficiency through finer-grained and customized support for the computing architecture, speedup improvement by 7.5x (ours) comparing to 4.3x (Blaze) in K-means. Although both HeteroYARN and Blaze are deployed to manage the FPGA accelerators in a heterogeneous cluster, accelerators in Blaze only accelerate the specific computation kernels, local sum of center distance calculation in K-means for example, while HeteroYARN adopts a finer-grained task-level mechanism in which FPGAs would participate more calculation processes to further reduce the computation overhead.

3 Heterogeneous Architecture Overview

We extend the native YARN platform for easily accessing FPGA accelerators on the CPU-FPGA cluster. Here, we first briefly describe the architecture of HeteroYARN framework, followed by our design of the extended resource management and task scheduler. Finally we introduce the mechanism of data sharing between CPU and FPGAs.

3.1 Overall Architecture

As shown in Fig. 2, a heterogeneous cluster deployed on HeteroYARN framework is composed of a cluster management node and several computing nodes integrated with FPGA accelerators. Nodes in the cluster communicate with each other through the Ethernet network. Computing nodes connect with FPGA devices through a medium driver. This driver provides a bi-directional data transfer and communication link between host and FPGA accelerators, and data is transferred in the form of data packets.

The distributed structure is utilized to create a heterogeneous environment, where the YARN framework constitutes the resource management infrastructure across the cluster. Computational tasks will be packed and assigned to FPGAs in the format they can recognize, so that most computational loads run on the FPGA accelerators. HeteroYARN employs the Direct Memory Access (DMA) engine to control the data transfer between CPU and hardware accelerators.

At present, the native Hadoop 2.0 platform only supports resource management of memory and CPU, so do the resource allocation and task scheduling. Thus, no interface for applications to use for access to hardware resources in the heterogeneous environment. In order to provide a generic system for usage of hardware resources, we modify resource-related modules in YARN, mainly involves RM and NM. We summarize the key modifications in the following sections.

As different kinds of applications run on heterogeneous clusters, more requirements also emerge. We extend the computing framework in YARN to provide interfaces for applying for and utilizing FPGAs and modify the subtask template for easy access to the hardware physical path when customizing the usage of accelerators for applications. In particular, we provide interfaces to set resource requirements for subtasks and specify the node label expression by the job client. Users are allowed to set the number and type of accelerator resources when they submit an application. The AM is extended to parse resource requests of the submitted job, which is obtained from the job configuration file. AM also needs to report the job label when registering this job to RM, and specify the number of CPU, memory and FPGA resources simultaneously for containers allocated during task running. The node, on which the subtask is located to facilitate accelerating tasks, obtains the physical path of allocated accelerator device by the Node Manager (NM).

3.2 Cluster Resource Management Extension

3.2.1 Resource Dimension Extension

YARN platform uses multi-dimensional vectors to represent different resources in the cluster. We expand the resource description dimension in account of integrated FPGA resources. Taking MapReduce framework for example, which contains two computing phases, Map and Reduce phases, we can extend the cluster resource representation to $< CPU, memory, MapAcc, ReduceAcc >$ when we need to deploy an accelerated MapReduce framework in the cluster. The dimension of the representation vector is related to the kind of computing framework and the number of processing steps involved, regardless of the specific algorithms running on the cluster.

3.2.2 Accelerator Resource Annotation

In order to differentiate FPGA accelerators that perform different computations, we mark nodes with the logical functionality of accelerators. For nodes containing FPGA accelerators, label annotates the kind of job and the computation

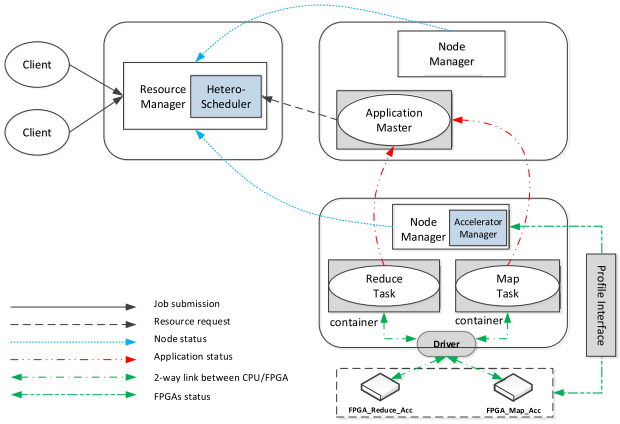


Fig. 2. Overview of HeteroYARN framework.

phase that it can run. In other words, instead of labelling accelerators just as "$Accelerator$", we propose to use representation like "$KMeansMapAcc, 2$", "$BayesRedAcc, 1$", where the number represents the amount of accelerators they have. This annotation way is useful to reduce the global placement overheads of an accelerated application. RM can group nodes with the same label together and separate the cluster into several partitions. Hence, the applications submitted with the specific label can only be scheduled to run on nodes that have the same label.

3.2.3 Multi-Dimensional Resource Scheduling

We propose HeteroScheduler, a resource scheduling algorithm for the heterogeneous cluster with FPGAs. This scheduler combines multi-dimensional Dominant Resource Fairness (DRF) algorithm and FIFO algorithm, to achieve better management and scheduling for multiple resources, like CPU, memory and FPGA accelerators (such as $MapAcc$ and $ReduceAcc$). HeteroScheduler uses a hierarchical queue to organize resource requests, and utilizes a tree structure to represent the cluster resources, where leaf nodes can be regarded as container requests. HeteroScheduler first selects the sub queue by DRF algorithm, and then determines the leaf queue for container requests using standard FIFO algorithm. Hierarchical structure is adopted to ensure the fairness of resource sharing in multi-tenant environments and mitigate the overhead of scheduling multiple jobs.

There are different groups of pending tasks in the HeteroScheduler. Tasks that request the same functional accelerators will be put together in one group. Resource allocation in the same group using FIFO strategy, which schedules tasks in accordance with the submitted order. In the different group, the scheduler uses the multi-dimensional DRF algorithm, which is an extension of the traditional DRF algorithm by incorporating the accelerator as dominated resources. It is also flexible to apply a priority scheduling as users can set the priority level of applications manually.

In brief, HeteroScheduler does not only ensure scheduling fairness and utilization of multiple resources, but also provides convenience for integration of other kinds of hardware accelerators in the future work.

3.3 Node Resource Manager Extension

We also extend the NM, which is responsible for resource virtualization, allocation and recovery of local containers.

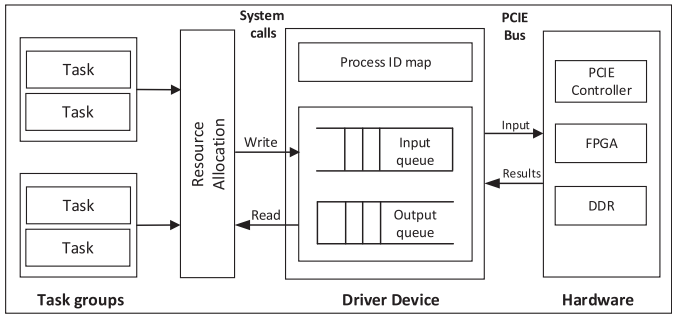


Fig. 3. Node accelerator manager design.

NM maintains the mapping of virtual accelerators to the physical file path of device. We provide a profile interface for users to adjust the number of virtual accelerators, CPU kernel amount, and the memory size. NM instance needs to report these information to RM at each startup. Nodes with Mapper and Reducer accelerators should report in the format $< CPU, MEM, MapAcc, ReduceAcc >$, and RM will configure these labels into YARN.

When receiving a request from AM to start the container with accelerators, NM will map the virtual accelerators to the physical location and register the container ID. Then, the corresponding host needs to select one of FPGA accelerators attached to this node for computing process. As shown in Fig. 3, computational data needs to be handled by the virtual device, queued up in the input buffer, and waits for being processed when the physical device is idle. The results will be returned to each process according to the unique process ID through output buffer. Then, NM reports the container status to RM via heartbeats while the cluster is in service. After a task is finished, NM will recycle resources occupied by the task, and report the idle resource margin to RM for scheduling. At present, the number of virtual devices needs to be adjusted experimentally according to the processing capability of the node. We will further study this open question on adaptive cluster configuration, which is similar to dynamic resource allocation by leveraging the workload information [39].

Correspondingly, the resource requests and container runtime interfaces provided by RM are also expanded to support the application that runs with accelerator cards. Considering the compatibility of the extended platform, native resource management interfaces are still retained in NM and RM, so that existing computing frameworks, such as native MapReduce, can run directly on the HeteroYARN platform without modification.

3.4 Bi-Directional Link Design

In our design, FPGA accelerator is mapped to a device file in the Linux system, and communicates with host applications using the file system call interface. The device driver can buffer the data transmitted between applications and accelerators, and coordinate the concurrent data transfer process. Applications use file system calls, read and write, to interact with the driver by blocking I/O.

In particular, the data buffers in the device driver include input buffer and output buffer, designed as a cyclic queue. Storage space of the buffer is a series circular address made up of contiguous memory in the system kernel. Each buffer block only stores one kind of data during one interactive

process, input data or computation results. The buffer block can be reused only when the data transfer is complete. Since DMA mode is used for communication control, hosts should send packet header and data separately to FPGAs, using a producer-consumer model. In the first stage, host relays the address of packet header and length of the packet to the accelerator and starts DMA process. In the second stage, the accelerator first pulls header message from the PC memory initiative and extracts the information about memory address of data and file format. Then the accelerator pulls the input data of the tasks to produce corresponding results and stores the results in the specified address which can be obtained from the message header. Finally, the accelerator informs the PC that the results have been generated through the interruption. During the transfer process, the previous stage of communication is initiated by the PC, the latter stage is processed by FPGAs, PC-side only needs to deal with the interrupt after task is completed. The Java Native Interface (JNI) approach is used to support this bi-directional data transfer link.

3.5 Dataflow of Accelerated Tasks

An accelerated task can be divided into two parts: the stub program in host side and the computation process in FPGA accelerators. Equally, the corresponding applications consist of software and hardware parts. In the software part, stub program in PC-side performs the subtask of data preprocessing and transfer. Stub program first converts input data into the format that FPGAs can process, and then send the data to allocated nodes with accelerators. Subsequently, FPGA accelerators start to execute the computation using parallel circuit. At the same time, the stub program monitors the computational status in FPGA, and fetches the de-serialization results immediately when computation is completed. Considering the different characters between accelerated algorithms and the range of total input amount for the whole task, there may be multiple times of data transfers between PC side and FPGAs during a single computing stage. For example, a Map phase may transmit more data than the Reduce phase (note: Combine stage is included in Map phase). The dataflow of an accelerated task in execution is given in Fig. 4. As shown in the figure, the input and output data of accelerated tasks need to be serialized and de-serialized respectively before they are transferred further. Thus, we provide related interfaces for developers to implement corresponding (de)serializing methods.

As the data processing flow changes, the AM, another module in Hadoop2.0, needs new interfaces to apply for FPGA resources when starting containers allocated with accelerators. The extended platform also provides functional interfaces for the job client to set FPGA resources required in the computing stages.

3.6 Compatibility of HeteroYARN

Computing frameworks, such as MapReduce, rely on the resource management interfaces provided by the platform to apply for and use the resources in the cluster. To handle the additional management requirements for FPGA resources, the corresponding management interfaces also should be added, which brings a problem of whether the extended platform continues to be compatible with the native framework. Therefore, we propose a compatible method at the protocol level after careful analysis of the communication

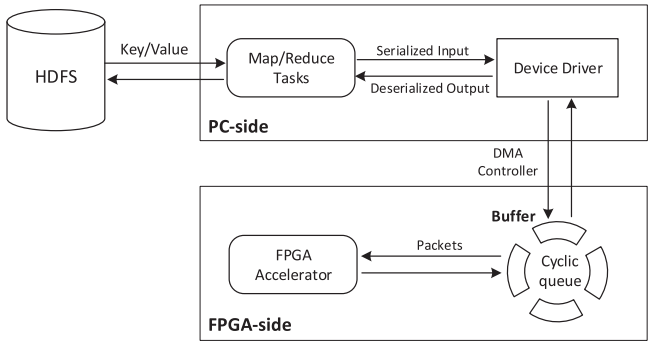


Fig. 4. Dataflow between PC and FPGAs during the life cycle of accelerated task.

protocol in native YARN and retain the native management interfaces so that the existing computing frameworks still can run on the HeteroYARN platform.

We propose a backward compatible serialization scheme to ensure the compatibility between HeteroYARN and the native framework. The underlying Remote Procedure Call (RPC) protocol for communication in the native YARN architecture utilizes the Google's Protocal Buffer (Protobuf) serialization framework to carry out message serialization. Protobuf provides a backwards compatible setting so that the version of both communication sides could be out of uniform, that the previous deserializer will directly ignore the newly added field. By setting the newly added attributes to be default value, using $'0'$ for example, resource requests from native computing framework can be expanded to more dimensions smoothly by the underlying serialization framework.

4 Heterogeneous Computing Framework

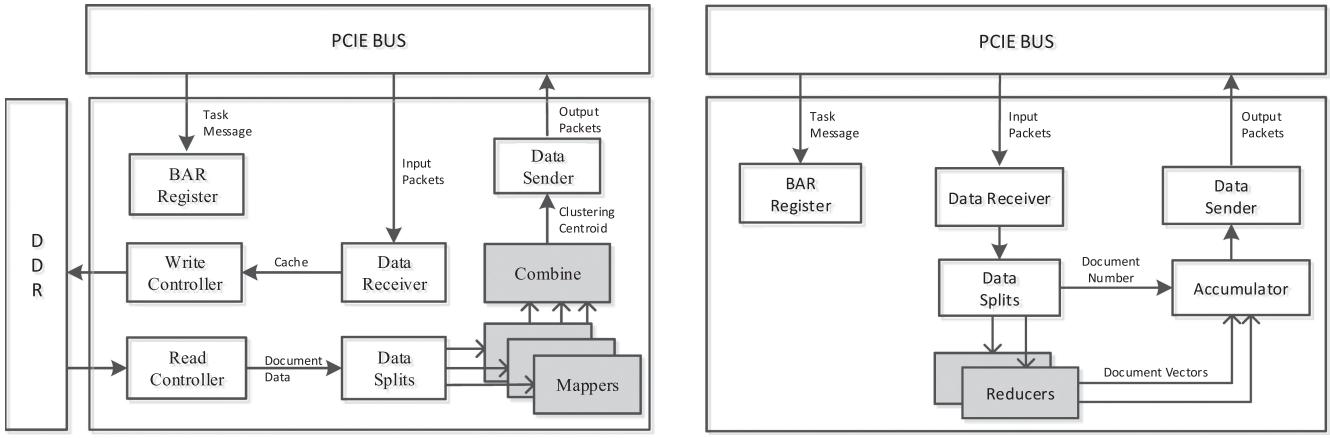
This section illustrates the effectiveness of the HeteroYARN. Contents include the workflow of HeteroYARN framework, the design of base interfaces to support programming applications with accelerators, and MapReduce implementations of K-Means and Naive Bayes classifier on HeteroYARN framework.

4.1 HeteroYARN Workflow

Applications running on a heterogeneous cluster can be categorized into two groups: ordinary applications and accelerated applications. When a user submits a job to HeteroYARN, the framework will first determine whether it chooses to accelerate the job using FPGAs or not. If so, computational tasks of FPGA portion will be ported to accelerators via the driver device.

The MapReduce implementation using FPGAs is an extension of the original MapReduce on HeteroYARN platform, inheriting the data processing flow. We refer readers to a tutorial [40] for more details. The differences lie in the Map and Reduce phases. Most computations are actually completed on accelerators in the extended framework. Data processing inside FPGAs is shown in Fig. 5. The MapAccelerator is designed to assign the data objects to the closest CLUSTER, and ReduceAcelerator tries to recount the intra-cluster centroid of each CLUSTER.[1] In order to decrease the size of intermediate data and further ease the pressure of data

1. To distinguish from computer clusters, we use "CLUSTER" to represent the clustering group in K-means.



(a) MapAccelerator module (b) ReduceAccelerator module

Fig. 5. Overall block-diagram of the datapath architecture.

transmission, we also perform the Combine phase on FPGAs, which is another data-intensive task to accumulate vectors of documents that contains the same centroid.

To build a generic system that supports MapReduce applications incorporated hardware FPGA resources, we divide the functionality of the hardware design into two parts: MapAccelerator module and ReduceAccelerator module. Fig. 5 presents the block-diagram of the overall design structure. As previously mentioned, the DMA controller is programmed to control data transfer via PCIE bus, namely DataReceiver and DataSender respectively. Data first get cached in Double Data Rate (DDR) to prevent the time loss caused by different data processing speed of computing kernels. The Bar Register provides interfaces to realize the read-write logical function of the internal register. If control module notices any map core is idle, the Bar Register would give a read command to the DDR, then data is fetched and calculated while the transmission of PCIE data is not blocked. Directed line segments in the diagram have shown the work flow in detail, and shaded boxes represent the user-specific hardware kernels. Depending on the underlying infrastructure, we can implement various applications not limited to MapReduce.

4.2 Interface Design

We also design and implement a wide range of interfaces for easy programming on the HeteroYARN framework, to make it easier to program applications using FPGAs. Our interfaces hide details about FPGA accelerator initialization and hardware programming by providing a set of development interfaces in JAVA. For example, the input and output data can be obtained by simply calling the AcceleratorReader and AcceleratorWriter. No explicitly technical OpenCL buffer manipulation or hardware programming operation is necessary for application programmers.

Developers can process arbitrary data types they want by overriding the medium driver and implementing serialization and de-serialization methods. We also provide the developers with setting interface to customize the map and reduce accelerated processing conveniently.

There are also interfaces for users to set the job label and resource demand when submit it, and users are allowed to specify the PC-side stub program to coordinate with FPGAs. HeteroYARN framework provides base classes, *AcceleratedMapper* and *AcceleratedReducer*, to ensure the key/value pairs in MapReduce can be utilized on FPGA accelerators directly. Developers should overwrite these requisite functions when they program to process data with the accelerator interfaces. Table 1 lists significant programming interfaces and their brief descriptions.

TABLE 1
Programming Interfaces

Interface	*Brief Description*
setJobTag	set the label for the job
isMapperAccelerated	whether accelerate Map task
setAcceleratedMapper	set accelerated Map task
setMapAcceleratorNums	MapAcc number for Map task
isReducerAccelerated	whether accelerate Recuce task
setAcceleratedReducer	Set accelerated Reduce task
setReduceAccelaratorNums	ReduceAcc number for the Reduce task
setMapSerializer	Map task data serialization class
setMapDeserializer	Map task data deserialization class
.....	.....

4.3 Computing-Intensive Algorithm Acceleration

On the HeteroYARN framework, we offload CPU-intensive tasks to the accelerators, and take K-means as the representative of iterative algorithms, Naive Bayes as the non-iterative one.

4.3.1 Data Format in Transmission

According to characteristics of the FPGA hardware processing, we intend to arrange data into a fixed message format that accelerators can process directly. The data exchange process can be described as follows: host splits the input data into a number of fixed-size data packets, then sends the data splits to accelerators for further calculation, finally obtains the computing results corresponding to each packet and merges them together.

As to packet format design, each message contains a complete input unit for computing and no semantic association between each other. The self-described message format is used so that FPGA accelerators can complete the computation without any other dependency information. Accelerator does

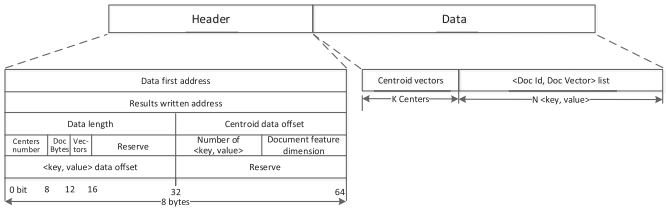


Fig. 6. Data format of accelerated K-means algorithm.

not need to care about which stub program of the computation message comes from, nor does it need to save the intermediate state on the accelerator. All stub programs that run the same type of computing task can share the same accelerators. Thanks to the high performance of FPGA hardware, accelerators may be waiting for input data most of the time during the sub-task execution. To alleviate this situation, we adopt multiple threads to process the communications between FPGAs and PC stub programs, a parallel way of sending and receiving data. Thus, data preprocessing tasks in host will not be suspended for receiving results from FPGA accelerators, which will reduce the time of waiting for data transmission between host and accelerators, so that host does not need to take up the accelerator in the data preprocessing phase.

For example, the input packets for K-means algorithm consist of header and data, as shown in Fig. 6. The data part contains K CLUSTER centroids and key-value pair inputs of Map tasks. The number of key-value pairs is determined by the size of the entire input packet and the size of each key-value pair. The packet header is a description of the computing tasks, including the type of each data field, the physical memory address for input data and result storage, and so on. In detail, the header mainly includes the start address and length of packet data in memory, the first address of results, the number of CLUSTER centroids and data offset, the number of key-value pairs and data offsets, the dimension of the CLUSTER feature vector and the size of the space occupied by each dimension in the vector. The output is also organized in the form of a packet, except that the file header is no more needed.

The driver detects whether the accelerator can accept input data using a polling technique. Once the state register is set ready, the host starts the DMA mode for data transfer. Accelerators notify the device driver by the interruption when transmission is complete or results arrive. If it is the completion of packet sending, the corresponding buffer block will be released. Or result arriving, the result buffer will be marked as ready and begin to wait for the running task to read or wake the blocked thread which waits for calculation results. In other words, the data packets are transferred by pulling between accelerators and host while handling message is transferred with the interrupt.

4.3.2 K-Means Algorithm Implementation

The K-means algorithm spends most execution time on calculating the distance between objects and updating centroids. Execution process involves a large number of floating-point multiplications and additions, causing computationally expensive on the CPUs. It is independent to compute the distance between each document with each centroid in the K-means algorithm, which means the process is easy to be paralleled. Hence, the K-means algorithm is a good candidate to be accelerated on FPGAs. Using the programming interfaces provided by HeteroYARN framework, we can program K-means algorithm implementation below. Part of the pseudo codes are shown in the Listing 1.

Listing 1. K-Means Map Phase (Pseudo Code in Java)

```
public class KMeansAcceleratedMapper extends
AcceleratedMapper <...> {
    // capturing mapTask, serializing the input to FPGA
    class KMeansMapAcceleratorWriter {...}

    // deserializing the result package from FPGA
    class KMeansMapAcceleratorReader {...}

    // centroid vector normalization
    void setup(Context context){
        ...
        while (reader.next(key, value)) {
        // vector normalization, ByteArrayWritable to
        FloatArrayWritable
          float model = 0;
          float[] array = new float[vector_dime];
          array = value.getFloatVector().getfloa-
          tArray();
          for(int i=0;i<vector_dime;i++){
             model += array[i]*array[i] ;
          }
          for(int i=0;i<vector_dime;i++){
             array[i] = Float.parseFloat(df.-
             format(array[i]/Math.sqrt(model)));
          }
        System.arraycopy(serialize(new Float-
        ArrayWritable(array)),...);
    }
    reader.close();
    super.setup(context);
  }
    // receiving data, serialization and deserialization
    void process(Context context){
    ...
        // send Thread, data to Driver
        sendTask=new Callable<String>(){
          ...
          writeFPGA(getCurrentKey(),CurrentValue
          ());
        }
        // receive Thread, object to YARN
        receiveTask=new Callable<String>(){
          ...
          ByteArrayWritable key=new ByteArray-
          Writable();
          KmeansMapOutByteArrayWritable value=new
          KmeansMapOutByteArrayWritable();
          finalContext.write(key,value);
        }
        service.submit(sendTask);
        service.submit(receiveTask);
  }
  ...
}
```

The Map phase completes two main tasks, first calculating the distance between each document vector and each CLUSTER center, and then classifying the document into

TABLE 2
Server Configuration Details

Node	CPU	Memory	I/O	Functional Accelerator
master	E5-2620V2	128G	2.4TB PERC H71 RAID5	none
slave1	E5-2650	128G	8T SAS RAID5 750EVO SSD	MapAcc ReduceAcc
slave2	E5-2650	128G	8T SAS RAID5 750EVO SSD	MapAcc ReduceAcc

the category, to which the nearest center belongs. The distance metric is euclidean distance, defined as follows:

$$x^2_{(d,c)} = \sum_{i=1}^{n} |d_i - c_i|^2, \tag{1}$$

where $d = (d_1, d_2, \ldots, d_n)$ and $c = (c_1, c_2, \ldots, c_n)$ are document vector and center presentation respectively. Both tasks are time-consuming processes and offloaded to Map accelerators. Moreover, we load the Combine process into Map accelerators to merge the intermediate key-values locally, aiming to reduce the amount of data transmission between FPGA accelerators and host. Combine process starts immediately when Map process completes. This process counts the number of documents and accumulates the document vectors in each CLUSTER. Then merged data are transferred back to host for further processing by Reducer.

The Reduce phase can be divided into two stages. First, it accumulates the global intermediate document vectors and the document number that assigned the same centroid. Second, it calculates the new centroid vector by dividing accumulated vector with the number of documents. Both the numerator and denominator are computed in the first stage. The first stage is executed on Reducer accelerators for parallel execution. The second stage is completed in CPU, as this task belongs to serial computing with little computational effort, only including k times vector decomposition during one iterative process.

4.3.3 Naive Bayes Classifier Implementation

As to Naive Bayes classifier implementation, we first train the raw data based on eigenvalue, in which two probabilities are computed: the prior probability $P(c)$ for each category c, and the conditional probability $P(t|c)$ for each word t in each category c. Then these values are sent to FPGAs in packets and persisted in the corresponding on-chip RAM. When the document data-points come, FPGA accelerators will first count the word frequency and the number of words of each document. Statistical results will be saved in the register. Finally, we obtain the probability $P(c|d)$ through the formula below:

$$P(c|d) = P(c) \prod_{1 \le k \le n_d} p(t_k|c), \tag{2}$$

where $P(t_k|c)$ is the probability of term t_k in category c, n_d is the length of the document d. Category with the maximum probability is the "best" category, to which document d belongs.

It is also worth noticing that map process is totally enough to find out the category for each document, which means that only the Map accelerator is needed for Naive Bayes classifier. The specific code implementation is similar to that of K-means.

5 Performance Evaluation

In this section, we first describe the hardware and software settings in the experiment. Then we evaluate the performance of computing-intensive tasks on HeteroYARN with different settings. We also analyze the experimental results to explore factors leading to performance bottleneck.

5.1 Experimental Settings

The experimental platform is a heterogeneous cluster consisting of 3 nodes, one manager, and two computing nodes. The computing node is integrated with FPGA cards using PCIE slots, and FPGA card contains a Xilinx XC7K410TFFG900-2 chip and 3.5 GB of on-board RAM. The data transfer between the host and accelerator cards is carried out through the PCIE bus using DMA mode, while the data description and control commands are transferred in PIO mode.

The HeteroYARN platform and MapReduce implementation are developed on the stable version of Hadoop 2.6.0, which first adopts label based scheduling. All nodes in the cluster have a 64 bit CentOS 6.5 operating system installed, and JDK1.7 is installed to support the operation of the Java application. The server configuration of the cluster is given in Table 2.

In the experiments, we take two representative computing-intensive algorithms, K-means and Naive Bayes, as the examples. The hardware portion of computation logic was programmed by hand in Verilog. We utilize the Vivado development tools to synthesize program into .Bit files, and then convert into corresponding .Mcs files, finally burn them into FLASH on FPGAs.

The training data for experiments are randomly generated, represented as a 128-dimensional vector for one document in text format, where each dimension is a single precision floating-point number, so that we can increase the size of dataset easily to hundreds of GB or even more.[2] The cosine distance is used to measure the vector similarity, and all input data are uploaded to the HDFS in advance.

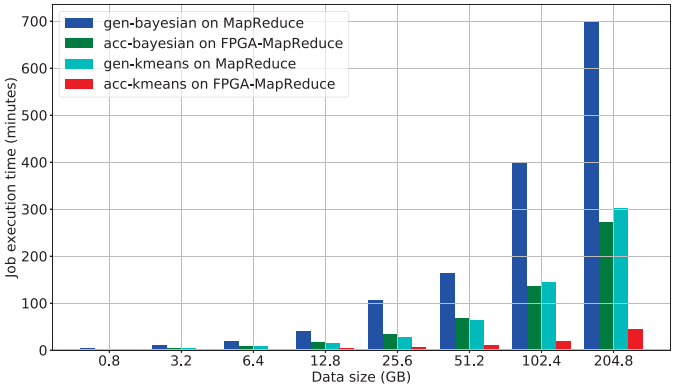
Note that there are some related studies on heterogeneous cluster using FPGAs. We do not compare HeteroYARN with them because: (1) their codes are not available; or (2) the issues we are concerning are not exactly the same.

5.2 Accelerating Performance

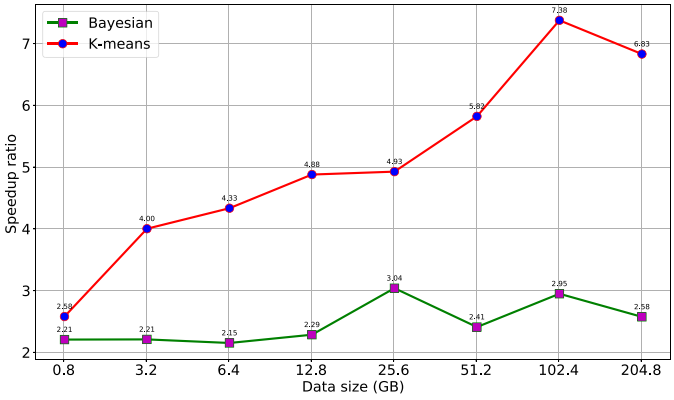
We first set the number of CLUSTERs to 16 to evaluate the acceleration performance with different data sizes. By taking the average of multiple experimental results to eliminate random errors, the final comparison of execution time is given in Fig. 7.

We can observe that the speedup ratio of accelerated applications in the heterogeneous cluster is closely related with data scales. As the trend in Fig. 7b shows, the speedup ratio is on the rise when data scales increase. To analyze the results, we monitor the job running in real-time and find that

2. We have uploaded part of the data to IEEE DataPort. Available: http://dx.doi.org/10.21227/fbpn-s032.



(a) Performance of Naive Bayes and K-means on HeteroYARN



(b) Speedup of accelerated job at different data volumes

Fig. 7. Accelerating performance.

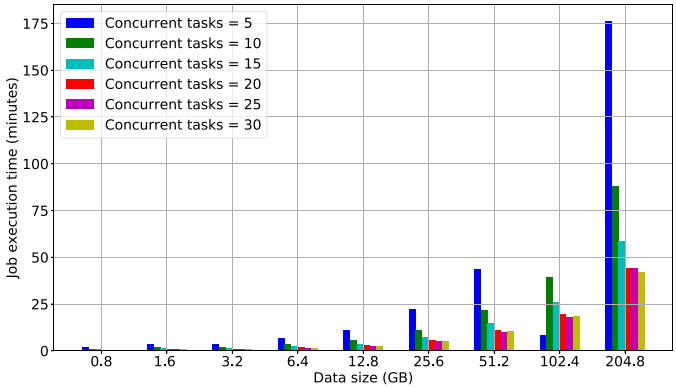


Fig. 8. The execution time with different concurrent tasks.

preparation process from job submission to startup takes up a certain amount of execution time. The proportion of the time spent on calculation phases rises with the data size increasing. Hence, the advantage of FPGA accelerators in calculation becomes more and more obvious. The implementations of K-means and Naive Bayes using FPGA accelerators achieve 7.5x and 2.3x speedup respectively, which demonstrates the effectiveness of HeteroYARN framework to accelerate computing-intensive algorithms.

We further analyze the reasons for different performances between K-means and Naive Bayes acceleration implementations. Experimental results indicate that the K-means algorithm performs better and achieves higher speedup ratio. In K-means instance, the operations offloaded to FPGAs include distance calculation and CLUSTER assignments in Map phase and centroids updates in Reduce phase. In Naive Bayes classifier, FPGAs accelerate the process of word frequency statistics and conditional probability product of each document. Obviously, the accelerators participate in more calculation processes in K-means implementation. On the other hand, the processes in K-means are more computing-intensive operations, such as floating-point multiplications. Thus, the acceleration effect is more obvious in the K-means implementation. On other words, the higher acceleration ratio an application has, the better efficiency will be achieved.

5.3 Multi-Job Comparison Analysis

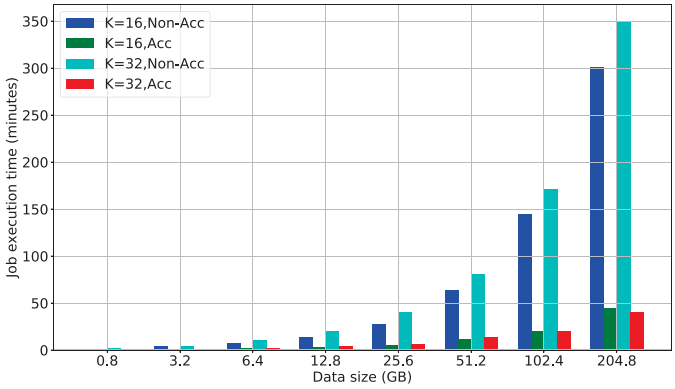
To evaluate the effectiveness with different number of concurrent tasks, we modify the cluster configuration to adjust the maximum number limit of concurrent tasks in each node (we set the limit as a fixed value 20 in other comparison experiments). Each concurrent task is an independent Map task or Reduce task to process a single data split. This setting is used to limit the amount of tasks initiated simultaneously on one node. Taking K-means algorithm implementation as an example, the experimental results are shown in Fig. 8. It is easy to find that the end-to-end job execution time has significantly shortened when the maximum number of concurrent tasks approaching around 20. Hence, the number of concurrent tasks should be set according to the configuration of computing nodes to get a better performance.

We also conduct experiments by adjusting the number of CLUSTER centers to 32 while other settings remain unchanged. In the K-means algorithm, the time complexity is $O(m \cdot n \cdot k)$ in one iteration, where m is the dataset size, n is the vector dimension, and k is the center number, so that computational complexity will increase exponentially when the number of CLUSTER centers increases by two times. The experimental results with different time complexity are shown in Fig. 9a.

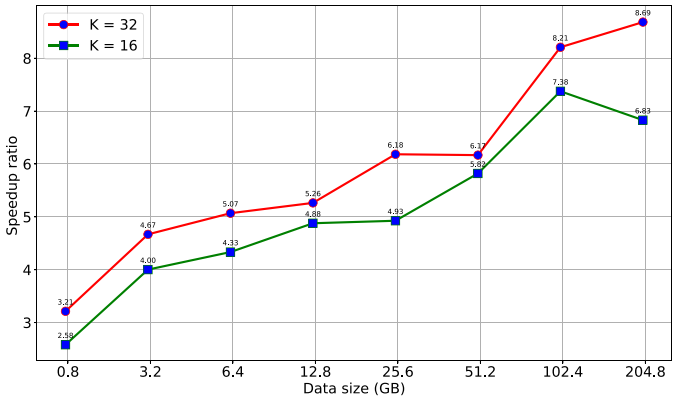
As shown in Fig. 9a, the job execution time is almost unchanged when increasing the number of the CLUSTER from 16 to 32 in the accelerated applications, while time increases obviously in non-accelerated ones. This is because computational components in FPGA logic will increase with the number of CLUSTER centers, remaining the computation time unchanged. However, the internal structure of CPUs are fixed, so the calculation time prolongs with calculation instructions increase. It can be concluded that the higher computational complexity of the job, the more obvious effect of FPGA accelerations. The speedup ratio curves up with data size increasing, shown in Fig. 9b, where K is the number of CLUSTER centers. It is interesting to find out that more CLUSTERs will lead to an increase in speedup ratio, since the complexity has greater impact on general processors.

5.4 Performance Evaluation of Resource Scheduling

In a heterogeneous cluster deployed on HeteroYARN framework, the computing nodes can be divided into two types according to whether FPGA accelerators are integrated into the node. For accelerated applications, the resource scheduling will only be really triggered when the RM receives the heartbeat of the nodes integrated accelerators, which means, the periodic communications between NM and RM. As a



(a) Performance comparison with different numbers of CLUSTER centers



(b) Speedup ratio with different data sizes

Fig. 9. Performance of different job.

result, it might increase the waiting time of an accelerated application for scheduling. Since it is difficult to count the scheduling time, we adopt the end-to-end execution time for performance comparison in the heterogeneous cluster, to reflect the scheduling capability indirectly. The size of the input data should be greater than 64 MB, the default block size, and the data copies are stored uniformly on nodes. Fig. 10 shows the results with different data sizes.

No matter the scheduling process of an application on a homogeneous or heterogeneous cluster, it includes job submission, first Container assignment, task start-up, task resource request, scheduling and task execution, along with disk IO overhead. Thus, it is not easy to compare the

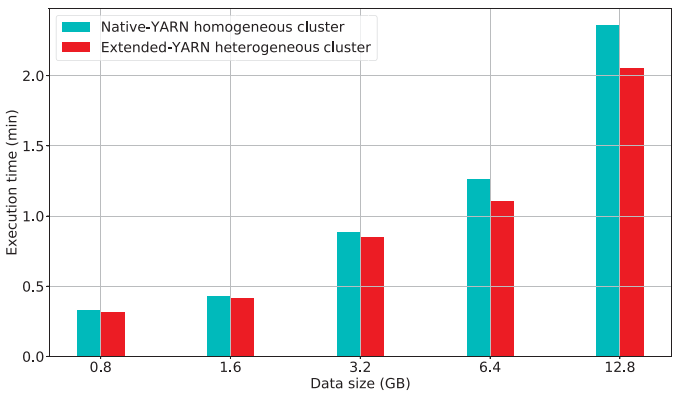


Fig. 10. Scheduling performance comparison with different data sizes.

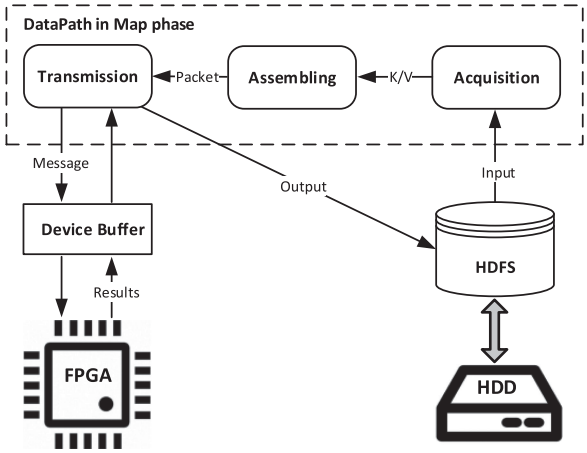


Fig. 11. Datapaths inside the accelerated task.

resource scheduling performance directly. We have to compare the end-to-end execution time for performance comparison. As shown in Fig. 10, the extended-YARN heterogeneous cluster performs better with less execution time, and the gap is increasing with the increment of data size. It is obvious that the heterogeneous resource scheduling has more advantages at larger scale data in the current experimental environments.

5.5 Performance Bottleneck Analysis

In order to find the key factor that affects the performance of accelerated jobs, we experimentally analyze the computational throughput of each concurrent module. Actually, taking the K-means algorithm as an example, the Map phase consumes more time than the Reduce phase, no matter with FPGA acceleration or not. Hence, our bottleneck analysis will mainly focus on the Map phase.

We analyze the data throughput of each concurrent module further in the Map task: acquiring input data, assembling key/value pairs into packets, sending data to accelerator and fetching results. As shown in Fig. 11, these three phases are designed as independent modules running in a serial manner. "Acquisition" module reads raw data from HDFS and parses them as key/value pairs; then Assembling module assembles the key/value pairs as packets; finally the packets are sent to the FPGA devices via "Transmission". On the setting that cluster number is 16 and data size is 102.4 GB, the data throughputs of each module in the accelerated Map task is shown in Table 3. There is a large gap of processing speed between different modules. Hence, we can conclude that the acquisition process of the input data is the bottleneck that restricts the performance.

Moreover, the acquisition process of the input data consists of two stages: reading the input data from HDFS, and de-serializing the input data to generate key/value pairs. In order to analyze the different effects of the two stages, we experimentally evaluate the data throughputs of reading data blocks

TABLE 3
Data Throughputs of Each Module

Module name	input acquisition	packets assembling	data transmission & result receiving
Throughput	89.6 MB/s	2660 MB/s	1000 MB/s

from HDFS and generating the key/value pairs. In the case that the number of concurrent processes is 20, the average throughput of reading HDFS data blocks is 250 MB/s, while the throughput of generating key/value pairs is only 89.6 MB/s. There is about 3 times gap between these two stages. That is, the key/value pairs parsing process greatly limits the processing speed of data acquisition, and further slows down the overall performance of the Map phase.

6 Conclusion

In this paper, we propose a heterogeneous FPGA architecture HeteroYARN for unified marking, managing and scheduling FPGAs resources in a heterogeneous cluster, and present the design of MapReduce using FPGAs, a hybrid implementation that integrates accelerators based on HeteroYARN framework. We explore how to program accelerated applications on the HeteroYARN framework, and evaluate it with the performance of K-means and Bayesian algorithm implementations. Our work has the following conclusions. (1) Our HeteroYARN is an efficient general-purpose runtime system to support the resource management and task scheduling with higher utilization in the heterogeneous cluster. (2) HeteroYARN framework abstracts the programming model to provide unified interfaces of applying for and utilizing FPGAs for users when developing applications. (3) The heterogeneous FPGA cluster can significantly accelerate computing-intensive data mining applications (speedup to 7.5x and 2.3x), represented by the K-means clustering and Naive Bayes classification respectively, and the processing pattern of the original framework has been inherited. All above works demonstrate an effective architecture that supports a generic heterogeneous FPGA cluster management based on YARN.

In the future work, we will provide more interfaces that support the easy and efficient access of FPGA accelerators, as well as other kinds of accelerators like GPU, for applications on Apache Spark and other programming models based on our heterogeneous cluster management framework. We will also evaluate performance of the latest SSD storage, to find the effective ways to alleviate performance limitations caused by I/O bottleneck in the heterogeneous cluster.

Acknowledgments

The authors are grateful to three anonymous reviewers for their comments and suggestions. This work is supported by the National Key Research and Development Program of China under grants 2016YFB0800402 and 2016QY01W0202, National Natural Science Foundation of China under grants U1836204, 61572221, 61433006, U1401258, 61572222 and 61502185, Major Projects of the National Social Science Foundation under grant 16ZDA092, and Guangxi High level innovation Team in Higher Education Institutions Innovation Team of ASEAN Digital Cloud Big Data Security and Mining Technology.

References

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, pp. 171–209, 2014.

[2] H. Esmaeilzadeh, E. R. Blem, R. S. Amant, et al., "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 365–376.

[3] N. Bharill, A. Tiwari, and A. Malviya, "Fuzzy based clustering algorithms to handle big data with implementation on apache spark," in *Proc. IEEE 2nd Int. Conf. Big Data Comput. Service Appl.*, 2016, pp. 95–104.

[4] I. Stoica, D. X. Song, R. A. Popa, et al., "A Berkeley View of Systems Challenges for {AI}," *CoRR*, vol. abs/1712.05855, 2017, http://arxiv.org/abs/1712.05855

[5] A. Putnam, A. M. Caulfield, E. S. Chung, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, 2014, pp. 13–24.

[6] A. Alhamali, N. Salha, R. Morcel, et al., "FPGA-Accelerated hadoop cluster for deep learning computations," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2015, pp. 565–574.

[7] B. Falsafi, B. Dally, D. Singh, D. Chiou, et al.,"FPGAs versus GPUs in data centers," *IEEE Micro*, vol. 37, no. 1, pp. 60–72, Jan./Feb. 2017.

[8] V. V. Kindratenko, J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W. W. Hwu, "GPU clusters for high-performance computing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2009, pp. 1–8.

[9] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun, "Energy-efficient acceleration of big data analytics applications using FPGAs," in *Proc. IEEE Int. Conf. Big Data*, 2015, pp. 115–123.

[10] K. H. Tsoi and W. Luk, "Axel: A heterogeneous cluster with FPGAs and GPUs," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2010, pp. 115–124.

[11] A. M. Caulfield, E. S. Chung, A. Putnam, et al., "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–13.

[12] V. K. Vavilapalli, A. C. Murthy, C. Douglas, et al., "Apache hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, 2013, pp. 1–16.

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, 2004.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX Conf. Hot Topics Cloud Comput.*, 2010, pp. 10–10.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, et al., "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 295–308.

[16] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proc. Eur. Conf. Comput. Syst.*, 2013, pp. 351–364.

[17] A. Verma, L. Pedrosa, M. Korupolu, et al., "Large-scale cluster management at Google with Borg," in *Proc. Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.

[18] Apache Hadoop, "The Apache Software Foundation" 2018. [Online]. Available: http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/

[19] A. Leeper, "MapR 5.0 Documentation : Label-based Scheduling for YARN Applications," 2015. [Online]. Available: http://doc.mapr.com/display/MapR/Label-based+Scheduling+for+YARN+Applications

[20] P. Cnudde, "Large scale distributed deep learning on hadoop clusters." 2015. [Online]. Available: http://yahoohadoop.tumblr.com/post/129872361846/large-scale-distributed-deep-learning-on-hadoop

[21] J. Zhu, J. Li, E. Hardesty, H. Jiang, and K.-C. Li, "GPU-in-Hadoop: Enabling MapReduce across distributed heterogeneous platforms," in *Proc. IEEE/ACIS 13th Int. Conf. Comput. Inf. Sci.*, 2014, pp. 321–326.

[22] S. Niu, G. Yang, N. Sarma, et al., "Combining hadoop and GPU to preprocess large affymetrix microarray data," in *Proc. IEEE Int. Conf. Big Data*, 2014, pp. 692–700.

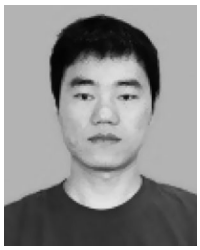
[23] A. Sabne, P. Sakdhnagool, and R. Eigenmann, "HeteroDoop: A MapReduce programming system for accelerator clusters," in *Proc. 24th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2015, pp. 235–246.

[24] M. Grossman and V. Sarkar, "SWAT: A programmable, in-memory, distributed, high-performance computing platform," in *Proc. 25th ACM Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2016, pp. 81–92.

[25] C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "GFlink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1275–1288, Jun. 2018.

[26] Y. K. Choi and J. Cong, "Acceleration of EM-based 3D CT reconstruction using FPGA," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 3, pp. 754–767, Jun. 2016.
[27] C. Zhang, P. Li, G. Sun, Y. Guan, et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2015, pp. 161–170.
[28] Z. Wang, B. He, and W. Zhang, "A study of data partitioning on OpenCL-based FPGAs," in *Proc. 25th Int. Conf. Field Programmable Logic Appl.*, 2015, pp. 1–8.
[29] J. Cong, M. Huang, D. Wu, and C. H. Yu, "Invited - heterogeneous datacenters: Options and opportunities," in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 16:1–16:6.
[30] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network FPGA clusters in a heterogeneous cloud data center," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 237–246.
[31] M. Owaida, D. Sidler, K. Kara, and G. Alonso, "Centaur: A framework for hybrid CPU-FPGA databases," in *Proc. 25th IEEE Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2017, pp. 211–218.
[32] M. Lattuada, F. Ferrandi, and M. Perrotin, "Data transfers analysis in computer assisted design flow of FPGA accelerators for aerospace systems," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 1, pp. 3–16, Jan.–Mar. 2018.
[33] K. Neshatpour, M. Malik, and H. Homayoun, "Accelerating machine learning kernel in hadoop using FPGAs," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 1151–1154.
[34] Y. Shan, B. Wang, J. Yan, Y. Wang, N.-Y. Xu, and H. Yang, "FPMR: Mapreduce framework on FPGA," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2010, pp. 93–102.
[35] Z. Wang, S. Zhang, B. He, and W. Zhang, "Melia: A MapReduce framework on OpenCL-based FPGAs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3547–3560, Dec. 2016.
[36] O. Segal, P. Colangelo, N. Nasiri, Z. Qian, and M. Margala, "SparkCL: A unified programming framework for accelerators on heterogeneous clusters," *CoRR*, vol. abs/1505.01120, 2015, http://arxiv.org/abs/1505.01120
[37] C. YuTing, C. Jason, F. Zhenman, L. Jie, and W. Peng, "When spark meets FPGAs: A case study for next-generation DNA sequencing acceleration," in *Proc. IEEE 24th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2016, pp. 64–70.
[38] M. Huang, D. Wu, C. H. Yu, Z. Fang, et al., "Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale," in *Proc. 7th ACM Symp. Cloud Comput.*, 2016, pp. 456–469.
[39] Y. Yao, J. Wang, B. Sheng, C. C. Tan, and N. Mi, "Self-adjusting slot configurations for homogeneous and heterogeneous hadoop clusters," *IEEE Trans. Cloud Comput.*, vol. 5, no. 2, pp. 344–357, Apr.–Jun. 2017.
[40] K. Kambatla, W. Y. Poon, and V. Srivastava, "Apache hadoop YARN workflow." 2014. [Online]. Available: http://blog.cloudera.com/blog/2014/05/how-apache-hadoop-yarn-ha-works/

Ruixuan Li received the BS, MS, and PhD degrees in computer science from the Huazhong University of Science and Technology, China, in 1997, 2000, and 2004, respectively. He is a professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. He was a visiting researcher with the Department of Electrical and Computer Engineering, University of Toronto from 2009 to 2010. His research interests include cloud and edge computing, big data management, and distributed system security. He is a member of the IEEE and ACM.



Qi Yang received the BS degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2014. He is working toward the PhD degree in the School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include big data analytic and data mining.

Yuhua Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2006. She is an associate professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. Her research interests include data mining, machine learning, and big data analysis. She is a senior member of the China Computer Federation (CCF).

Xiwu Gu received the PhD degree in computer science from the Huazhong University of Science and Technology, in 2007. He is an associate professor of the School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include distributed system, big data, and middleware.

Weijun Xiao received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, China, in 1995 and 1998, respectively, and the PhD degree in computer engineering from the University of Rhode Island, in 2009. He is an associate professor with the Department of Electrical and Computer Engineering. His research interests include computer architecture, networked storage system, embedded system, and performance evaluation. He is a senior member of the IEEE and the IEEE Computer Society.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. He is also a distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyberphysical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published more than 630 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.