JCST

Vol.37 No.6 Nov.2022

ISSN 1000-9000(Print)
/1860-4749(Online)
CODEN JCTEEM

Journal of Computer Science & Technology

SPONSORED BY INSTITUTE OF COMPUTING TECHNOLOGY
THE CHINESE ACADEMY OF SCIENCES &

CHINA COMPUTER FEDERATION

SUPPORTED BY NSFC

CO-PUBLISHED BY SCIENCE PRESS &

SPRINGER

Li LL, Liu JY, Fan JP *et al.* SOCA-DOM: A mobile system-on-chip array system for analyzing big data on the move. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(6): 1271–1289 Nov. 2022. DOI 10.1007/s11390-022-1087-z

SOCA-DOM: A Mobile System-on-Chip Array System for Analyzing Big Data on the Move

Le-Le Li[1,2] (李乐乐), Jiang-Yi Liu[1] (刘江佾), Jian-Ping Fan[3] (樊建平), *Member*, *IEEE*
Xue-Hai Qian[4,5] (钱学海), *Member*, *IEEE*, Kai Hwang[6] (黄　铠), *Fellow*, *IEEE*
Yeh-Ching Chung[6] (钟叶青), *Senior Member*, *IEEE*, and Zhi-Bin Yu[1,*] (喻之斌), *Member*, *CCF*, *IEEE*

[1] *Center for Heterogeneous and Intelligent Computing, Shenzhen Institutes of Advanced Technology Chinese Academy of Sciences, Shenzhen 518055, China*

[2] *School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China*

[3] *Center for High Performance Computing, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences Shenzhen 518055, China*

[4] *Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles CA 90089-0001, U.S.A.*

[5] *Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0001, U.S.A.*

[6] *School of Data and Science, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China*

E-mail: {li.ll, jy.liu, jp.fan}@siat.ac.cn; xuehai.qian@usc.edu; {hwangkai, ychung}@cuhk.edu.cn; zb.yu@siat.ac.cn

Received October 20, 2020; accepted April 25, 2022.

Abstract Recently, analyzing big data on the move is booming. It requires that the hardware resource should be low volume, low power, light in weight, high-performance, and highly scalable whereas the management software should be flexible and consume little hardware resource. To meet these requirements, we present a system named SOCA-DOM that encompasses a mobile system-on-chip array architecture and a two-tier "software-defined" resource manager named Chameleon. First, we design an Ethernet communication board to support an array of mobile system-on-chips. Second, we propose a two-tier software architecture for Chameleon to make it flexible. Third, we devise data, configuration, and control planes for Chameleon to make it "software-defined" and in turn consume hardware resources on demand. Fourth, we design an accurate synthetic metric that represents the computational power of a computing node. We employ 12 Apache Spark benchmarks to evaluate SOCA-DOM. Surprisingly, SOCA-DOM consumes up to 9.4x less CPU resources and 13.5x less memory than Mesos which is an existing resource manager. In addition, we show that a 16-node SOCA-DOM consumes up to 4x less energy than two standard Xeon servers. Based on the results, we conclude that an array architecture with fine-grained hardware resources and a software-defined resource manager works well for analyzing big data on the move.

Keywords edge computing, mobile architecture, resource management, big data analytics, software-defined system

1 Introduction

Recently, a wide range of the requirements from spaceflight and aviation to terrestrial vehicle applications for analyzing big data on the move are booming. We name this computing paradigm as ADOM. For spaceflight applications, orbital edge computing was proposed in 2020 to improve the computational power of nano-satellite constellations [1]. For aviation applications, evaluating the health and diagnosing the faults of aircraft systems are getting increasingly popular [2–4]. Even for low-altitude applications, real-time video ana-

Regular Paper

This work was supported in part by the Key Research and Development Program of Guangdong Province of China under Grant No. 2019B010155003, the National Natural Science Foundation of China under Grant Nos. 61672511, 61702495, and 61802384, the Shenzhen Institute of Artificial Intelligence and Robotics for Society, The Chinese University of Hong Kong, Shenzhen, and the Alibaba Innovative Research Project for Large-Scale Graph Pattern Discovery, Analysis, and Query Techniques.

*Corresponding Author

lysis on drones is proliferating rapidly [5]. For terrestrial applications, big data is also gradually employed to evaluate the vehicle health, trying to avoid crash [6–8]. In addition, ADOM is projected to rapidly get even more pervasive in the near future.

This wide adoption "hope" of ADOM poses six new requirements for computing systems including hardware and system software. First, the hardware volume of such a computing system should be low enough to be deployed inside a moving object such as a nano-satellite (e.g., 10 cm×10 cm×10 cm) (R1). Second, the computing system should consume as little energy as possible (R2) because the energy capacity on moving objects is always limited. For example, nano-satellites usually employ solar energy to power their computing systems while the size of their solar panels is small, generating very limited energy. Third, the weight of such a computing system should be as light as possible for easy moving (R3). Heavier weight results in more cost to move the computing systems. Fourth, the hardware resource consumption of the resource manager of ADOM needs to be as little as possible (R4) because the hardware resource is very limited. Fifth, ADOM systems are expected to be high performance (R5) since sharply increasing amount of data needs to be analyzed on moving objects to provide more and new functional services. Last, systems used for ADOM should be highly scalable (R6) because in some spaceflight cases (e.g., nano-satellite applications), the form factor of the system can only be very small (e.g., 5 cm×5 cm×1 cm) while in terrestrial application cases (e.g., car, bus, and truck applications), the form factor can be relatively large (e.g., 30 cm×20 cm×1 cm).

However, these conflicting requirements (e.g., R5 conflicts with R1, R2, R3, and R4) are posing unprecedented challenges to existing computing systems such as cloud computing and edge computing platforms. Cloud computing platforms are obviously overqualified for R5 but fail to satisfy R1, R2, R3, and R4 for ADOM. It seems that edge computing platforms including fog computing [9–12], mobile cloud computing [13–15], and mobile edge computing [16, 17] systems are good candidates for ADOM. However, edge computing typically deploys one or several standard servers such as 1U (a unit specifies that the width of a server is 48.26 cm and the height is 4.445 cm) or 2U (the width is the same as that of 1U but the height doubles) servers closed to the network edge as computing platforms. Although these platforms are called "micro data centers" [18, 19], "data center in a box" [20], or "cloudlet" [15], they are large in size and heavy in weight and fail to meet R1 and R3 of ADOM. The size of a base station which is equipped with digital signal processors may be smaller than a standard 1U server, and so does the weight, but digital signal processors are not suitable for big data analysis.

To address these challenges, we propose a novel system named SOCA-DOM to support big data analysis on the move. It contributes four innovations from both hardware and software aspects.

First, it innovates a mobile system-on-chip array architecture by designing different sizes of Ethernet network communication boards. A number of homogeneous or heterogeneous mobile system-on-chips seat on a board and communicate with one another via the board. Two such boards can be connected by the Ethernet network and therefore many boards can communicate directly or indirectly. As such, this hardware architecture is highly scalable (for R6), energy-efficient (for R2), space-efficient (for R1), and light-weight (for R3), and can achieve high performance (for R5).

Second, we design data plane, configuration plane, and control plane for the resource manager to make it consume as little hardware resource as possible (for R4).

Third, SOCA-DOM contributes a two-tier software architecture for the resource manager. It employs the master-slave mode to manage the mobile system-on-chip nodes, but several master nodes can be configured to act as the slave nodes of a super-master if needed. The benefit is that a mobile system-on-chip cluster can be easily configured to a number of zones where different zones serve different types of applications (for R6).

Finally, we design a synthetic metric to more accurately represent the computational power of a system-on-chip node compared with the traditional metrics such as the memory size and the number of cores. The synthetic metric considers the fine-grained information about CPU cores such as the clock speed, CPU model, cache, and so on. Then we propose to schedule tasks by using this metric (for R5). Our experimental results show that this can improve the average performance of Spark programs and Flink programs by factors of 4x and 3x, respectively.

Putting it all together, we implement a prototype of SOCA-DOM. It can run all the Spark programs and Flink programs from the Hibench benchmark suite and meet all the requirements of ADOM.

The rest of the paper is organized as follows. Section 2 presents the background of ADOM and the motivation of this work. Section 3 describes the hardware and the software design of SOCA-DOM whereas Section 4 depicts its implementation. Section 5 presents the experimental methodology. Section 6 provides the experimental results and analysis. Section 7 discusses the limitations of SOCA-DOM. Section 8 describes the related work. Section 9 concludes the paper.

2 Background and Motivation

2.1 Big Data Analysis on the Move

ADOM is tightly related to edge computing, and we therefore introduce edge computing first. Currently, edge computing paradigm has been proposed to address the issues such as non-real-time operations occurred in cloud computing. The key idea is to move the computing platform from a cloud center to the edge of networks where the data is generated. Edge computing has been used in a wide range of applications including orbital edge computing[1], smart home[21], self-driving cars[22], real-time video analysis[23], online gaming[24, 25], and so on. Depending on various applications, the requirements such as computational power and hardware form factor for edge computing platforms are significantly different. For example, the real-time video analytics may need powerful GPUs[23] while wimpy mobile processor cores are acceptable[26] for other applications such as MapReduce style big data processing.

ADOM is a special kind of edge computing since it employs the same idea that the data is processed on the edge but with a unique feature: "analyzing data on the move". This feature makes that the six requirements, including low volume, low energy consumption, low hardware resource consumption, light in weight, high performance, and high scalability of the ADOM computing platform, must be satisfied at the same time. Other edge computing applications such as the smart home do not necessarily need to meet all the requirements at the same time because the computing platforms are quiescent and in turn the platform weight is not concerned much.

There is a wide range of booming requirements for ADOM from spaceflight and aviation to terrestrial applications. For example, there is an exponential growth in nano-satellite launches over the past two decades and commercial satellite constellations today typically consist of hundreds of Earth-observing and camera-equipped nano-satellites, generating a huge amount of data[27]. However, due to the weak computational power on the satellites, all the data needs to be sent back to Earth to process, which causes a 5.5 h delay on average before the data reaches customers[1]. As such, it is hard to employ current computing platforms of satellites to implement the concept of space-as-a-service[27]. For terrestrial applications, monitoring the big data of the key components of a car is proposed to be analyzed online to avoid crash in advance[8].

2.2 Motivation

2.2.1 *Limitations of Existing Hardware Platforms*

Existing cloud computing platforms typically consist of hundreds of thousands of servers, which are extremely energy hungry and occupy huge space. Even a single standard server (e.g., 1U or 2U server) is bulky and cannot be moved easily. This indicates that even a single standard server is ill-suited for building ADOM platforms because it cannot meet the requirements of low volume (R1), low energy consumption (R2), light in weight (R4), and high scalability (R6) for ADOM. Moreover, it also implies that existing edge computing platforms are not suitable for ADOM because they employ one or several standard servers.

On the other hand, the computational power of a modern mobile system-on-chip is increasing sharply, which provides a great opportunity for building ADOM platforms. For example, a product of Samsung's mobile system-on-chip named Exynos 2100 could have up to 3.0 GHz CPU frequency based on ARM Cortex-A78. However, the performance of a mobile system-on-chip is still not enough to run big data applications such as Apache Spark programs because of the limited memory capacity and CPU cores of a single mobile system-on-chip. We need to organize a number of mobile system-on-chips in a space-efficient and scalable manner to meet the demand of high performance (R5) for ADOM. This motivates us to design a new hardware architecture for ADOM.

2.2.2 *Resource Management for ADOM Platforms*

Since the hardware resource of a system-on-chip cluster is still limited, the resource manager should consume as few resources as possible to meet the requirement of low hardware resource consumption (R3). Moreover, the resource manager should be flexible enough to adapt to different uses cases. For example, the mobile system-on-chip nodes of ADOM may need to be dynamically organized into different zones for different types of applications. Hence, a number of zones with no applications can be turned off to save energy to achieve the low energy consumption (R2) for

ADOM. However, these requirements cannot be satisfied by existing resource managers such as Mesos[28] and YARN[29] for big data analysis clusters, because these managers themselves consume a lot of resources such as CPU and memory which are limited on ADOM platforms. This motivates us to develop a "software-defined" resource manager for ADOM platforms with several important energy-efficient designs for big data workloads.

3 SOCA-DOM Architecture

3.1 Overview

SOCA-DOM consists of two parts: a mobile system-on-chip cluster (hardware) and a resource manager named Chameleon (software). The cluster employs a mobile system-on-chips array architecture where the mobile system-on-chips seat on a communication board. Chameleon is designed to manage the resources of an ADOM system. The node management of Chameleon innovates a two-tier architecture which is highly scalable. We design data plane, configuration plane, and control plane for Chameleon to make it "software-defined".

3.2 Hardware Design

The hardware design of SOCA-DOM should be highly scalable, energy-efficient, space-efficient, and light in weight at the same time. To achieve this goal, we employ "disaggregated architecture" but with different levels. The key idea of the disaggregated architecture is to break monolithic servers into disaggregated, network-attached hardware components such as CPU, DRAM, and disks. Since each hardware component can operate or fail on its own and its resource allocation is independent from other components, hardware resource disaggregation brings many benefits such as greatly-improved scalability, resource utilization, elasticity, heterogeneity, and failover.

The disaggregated architecture is not a single unified architecture. Instead, it has a number of variants with different levels. For example, the Open Computer Project (OCP)①, Intel Rack Scale Architecture (RSA)②, HP "the machine"③, and the Berkeley Firebox[30] build or propose the system-level disaggregated architecture. Other levels include disaggregated memory[31–33] and disaggregated flash[34, 35].

The key enabler of the disaggregated architecture is the network technology. Different levels of hardware disaggregation have different requirements for the performance of networks. For example, the latency between CPU and remote flash memory via networks can be approximately equal to the access latency of CPU to local flash memory by sophisticated software optimization[35]. However, the access latency between CPU and local DRAM is much shorter than that between CPU and remote DRAM via networks. Despite the recent advances in the network technology④, it is still difficult to make the access latency of remote DRAM as short as that of local DRAM.

For this reason, we do not employ memory disaggregation in our hardware design. Instead, we innovate an architecture where an array of system-on-chips are installed on a compact network communication board. Fig.1 illustrates the layout of the communication board. As can be seen, the network components such as network switch and scheduler are integrated in the board, performing the communication function between the on-board system-on-chips. We design a number of bayonet-type slots on each board which can install mobile system-on-chips. The bayonet-type slot design makes it easier to replace a system-on-chip if it is damaged. Each mobile system-on-chip integrates an ARM mobile CPU, a mobile GPU, memory, and rich interfaces such as HDMI, USB, and SDIO. The DS (Debug Console) beside each CPU in Fig.1 is an interface for debugging. For each system-on-chip, we also design a USB3.0 interface, a secure digital (SD) memory interface (TF-TransFlash), and an NVME SSD interface for convenient data storage.

We provide three circuit diagrams of the communication board to illustrate its structure further. Due to the limited space, we only introduce the main components of the board. In Fig.2, the main chip labeled with "217" is responsible for the exchanges of network data among the nodes on the board. Meanwhile, we have another chip labeled with "129" showed in Fig.3 which accounts for exchanging data to external connected devices. Finally, Fig.4 demonstrates the power manager system labeled in "73" to provide multiple stable voltages such as 0.9 V, 1.2 V, 1.5 V, and 3.3 V.

①https://www.opencompute.org, Aug. 2019.

②https://www.intel.com/content/www/us/en/architecture-and-technology/intel-rack-scale-architecture.html, Aug. 2019.

③https://www.hpl.hp.com/research/systems-research/themachine, Aug. 2019.

④https://www.intel.com/content/www/us/en/high-performance-computing-fabrics, Aug. 2019.

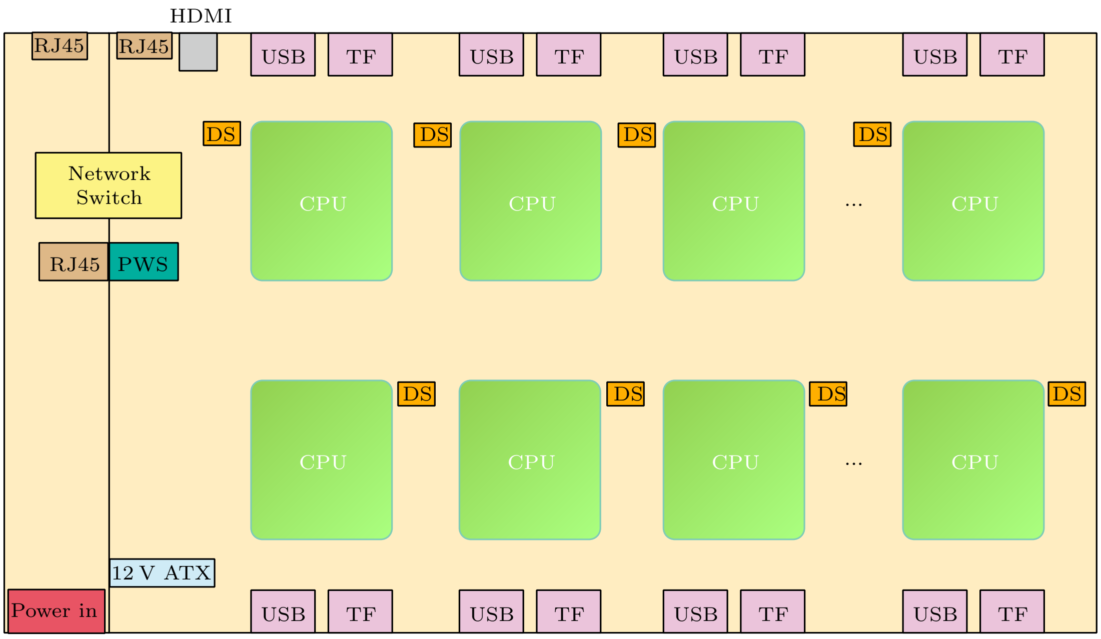


Fig.1. Layout of the network communication board for an array of mobile system-on-chips.

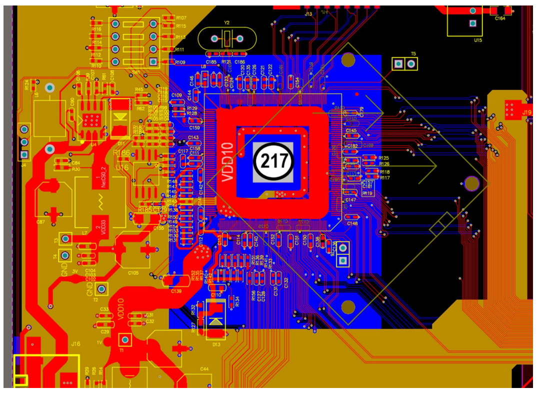


Fig.2. Main chip labeled with "217" for network data exchange of the communication board.

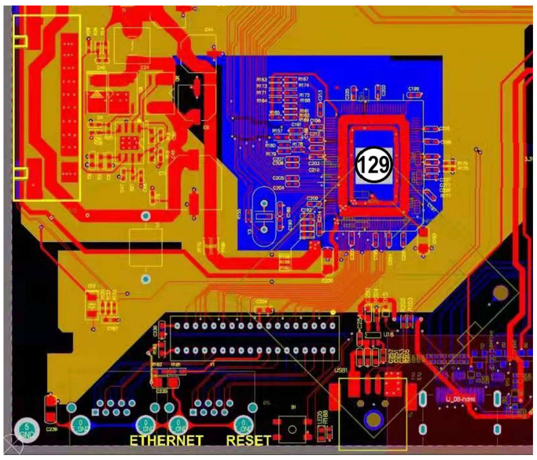


Fig.3. Main chip labeled with "129" for exchanging data to external connected devices of the communication board.

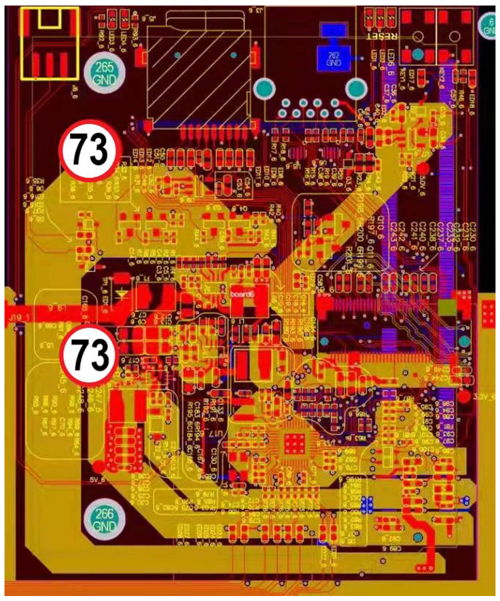


Fig.4. Power manager system labeled with "73" of the communication board.

We also design interfaces that are shared by the on-board system-on-chips. HDMI is used to connect one of the system-on-chips on a board to a screen. The 12 V ATX is used to supply power for the board and its components such as processors. Note that there are three RJ45 interfaces for the Ethernet network. The two RJ45 interfaces on the edge of a board are used to connect a board to another board or other computational nodes such as a traditional X86 CPU-based server by an Ethernet network cable. The other RJ45 interface is built in the form of RJ45 pins, connecting two boards directly in a compact way.

Our mobile system-on-chip array architecture ensures the scalability and flexibility of ADOM systems. For example, we can use the RJ45 pins on the board to directly connect a number of such boards to form a homogeneous ADOM cluster, as shown in Fig. 5. This is significantly more space-efficient than the Glasgow Raspberry Pi cloud which uses commercial routers on the shelf for communications and a Raspberry Pi mother board for periphery device IO for each processor[36]. Moreover, one or more boards can be connected to traditional x86-CPU based servers or others such as RISC-V CPU based servers, as shown in Fig.6.

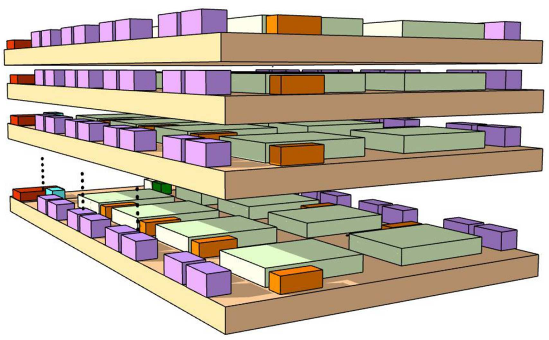


Fig.5. Illustration of an ADOM cluster.

3.3 Software Design

We now describe the design of our resource manager Chameleon with two primary design concerns: flexibility and lightweight.

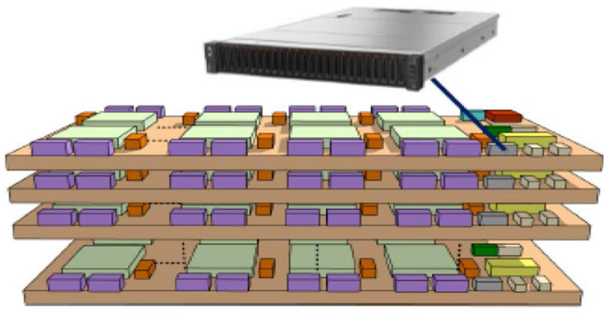
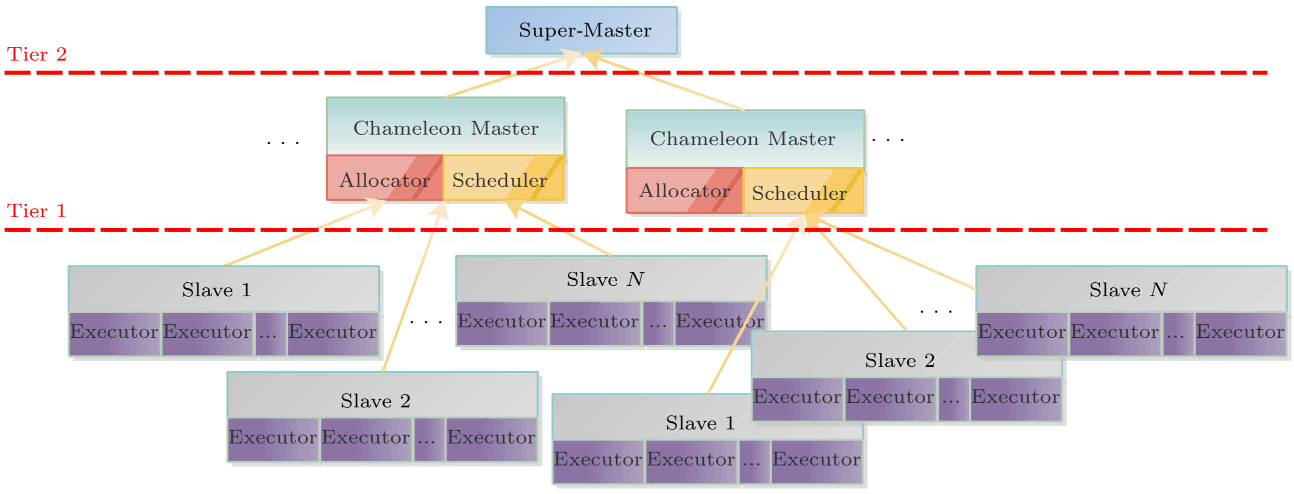


Fig.6. Heterogeneous clusters.

3.3.1 *Two-Tier Architecture*

We design a two-tier architecture for Chameleon as shown in Fig.7. As can be seen, Chameleon employs a master-slave mode but with two tiers. Moreover, the one- or two-tier architecture can be switched to each other smoothly. In tier 1, a computing node is selected as the master of a cluster to manage a number of other nodes which are called slaves. There is a daemon-called scheduler to passively collect the resources of the slaves and register the big data frameworks such as Apache Spark and Flink running on the cluster. There is another daemon on each slave node to manage the resources of the node and execute the assigned tasks. The daemon on the slave reports its resources to the master via heartbeats. In summary, the function in tier 1 is similar to that of other resource managers such as Mesos[28] and YARN[29].

In tier 2, another mobile system-on-chip node is added and acts as masters in tier 1. We call the master in tier 2 super-master. The masters in tier 1 report their resources to the super-master via heartbeats. More-


Super-Master
Tier 2
Chameleon Master
Allocator
Scheduler
Chameleon Master
Allocator
Scheduler
Tier 1
Slave 1
Executor Executor ... Executor
Slave N
Executor Executor ... Executor
Slave 2
Executor Executor ... Executor
Slave N
Executor Executor ... Executor
Slave 2
Executor Executor ... Executor
Slave 1
Executor Executor ... Executor
Fig.7. Two-tier architecture of Chameleon.

over, the big data analytical frameworks are registered in the super-master when Chameleon is configured as the two-tier architecture. Fig.8 shows the detailed steps about how a big data analysis program is registered and scheduled to execute. In step ①, the masters register their information on the super-master. In step ②, the drivers of big data analysis frameworks such as Apache Spark and Flink register their information on the super-master. In step ③, the super-master decides which zone for a big data analysis program to run and passes this information to the corresponding driver. Finally, the driver registers on the master of the assigned zone and the master schedules the corresponding tasks to run on the slaves of the zone, as shown in step ④.

3.3.2 *"Software-Defined" Design*

On how to make Chameleon adaptive to the extremely diverse situations of ADOM applications, we are inspired by software-defined network (SDN)[37], but with our innovations. One of the SDN's pillars is the decoupling of the control plane and the data plane. We also decouple the control plane and the data plane in Chameleon, but we propose a configuration plane and decouple it from the other two planes, as shown in Fig.9. The reason is that big data analytical frameworks such as Apache Hadoop and Apache Spark have a large number (e.g., 40) of configuration parameters. Some of these parameters are critical to performance but different frameworks need different configuration optimization approaches for optimized performance. For example, the optimization approach used for optimizing the configurations of Hadoop programs[38] cannot achieve the optimized performance for Spark programs[39]. We therefore propose a configuration library containing a number of configuration optimization algorithms in the configuration plane and provide north and south interfaces for the other parts of Chameleon to program, as shown in Fig.9. Note that when a framework registers on Chameleon, its configuration parameters are set by Chameleon.

As shown in Fig. 9, our control plane is in charge of task scheduling. Similar to the configuration optimization, different types of programs may need different scheduling algorithms or polices to achieve optimal performance. We propose to build a scheduling algorithm library in the control plane. Moreover, we integrate a piece of interesting work, a time-space sharing scheduling abstraction[40], in Chameleon to further optimize the performance of big data analytical programs. Note that both configuration and control planes can allocate resources for tasks. However, the configuration plane performs a passive resource allocation because it allocates the resources for tasks before a program starts to run and cannot re-allocate the resources when the program is running. In contrast, the control plane can change the resource allocation even when a program is running and we therefore call it active resource allocation. The data plane is a central place in Chameleon to maintain the status and information of hardware resources. Similar to the control and configuration planes, north and south interfaces are provided for the data plane. As shown in Fig.9, the three planes can call each other's interfaces and all the interfaces can also be called by programs outside Chameleon.

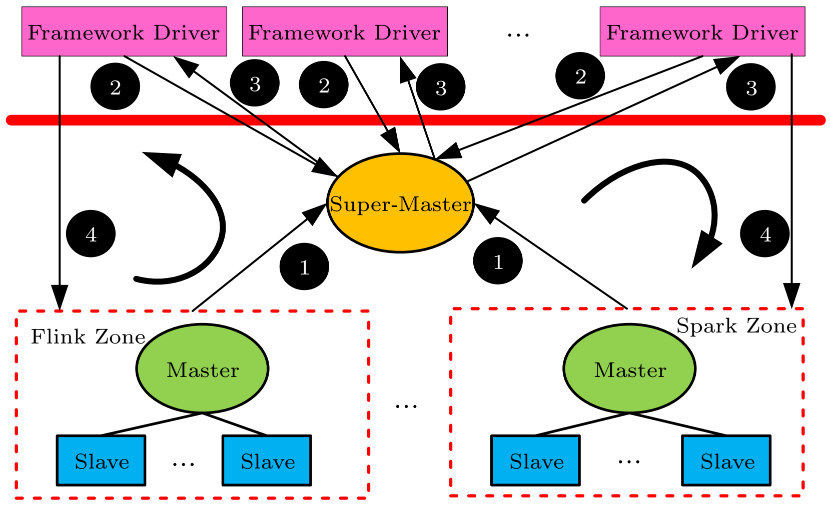


Fig.8. Workflow of big data analysis program registration, scheduling, and execution when Chameleon is configured as a two-tier architecture.

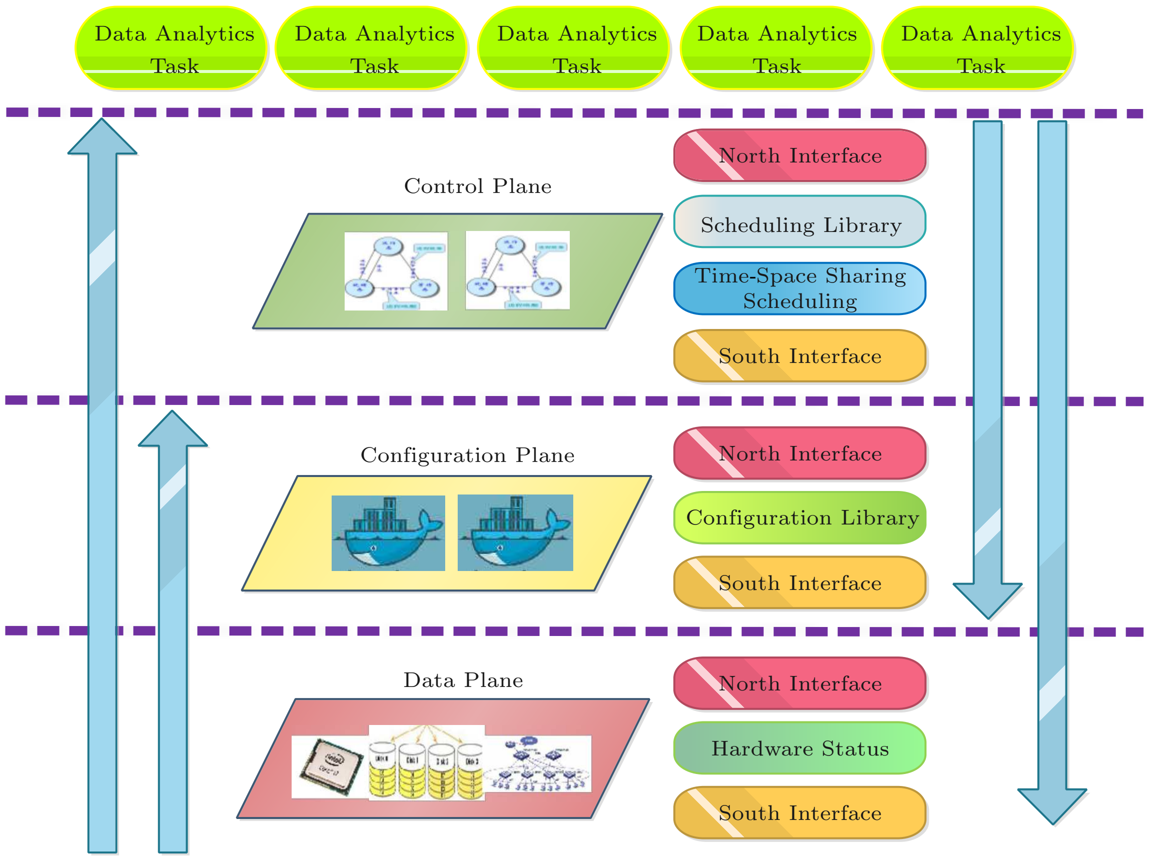


Fig.9. Three planes of Chameleon.

The reason for the "software-defined" resource manager is two-fold. First, we emphasize that the functions of the resource manager is programmable. Second, the data, control, and configuration planes are decoupled. These two aspects make the modules of Chameleon be loaded into memory on demand, saving CPU and memory resources.

Actually, the idea of "software-defined" software has attracted attention in recent years including software-defined cloud computing [41, 42] and software-defined operating systems [43]. However, these are early proposals while we implement a real software-defined resource manager for ADOM platforms.

3.3.3 *Fine-Grained Resource Metric*

As aforementioned, a mobile system-on-chip contains heterogeneous processors such as CPUs and GPUs, and heterogeneity is therefore an essential feature of ADOM systems. The resource manager should therefore consider fine-grained resource heterogeneity as much as possible when it schedules tasks. However, the popular resource managers mainly use the coarse-grained resource metrics such as the number of CPU cores and the size of memory to schedule tasks. For example, Mesos [28] offers resources for programs as a combination of CPU cores, free memory capacities, and remaining disk capacities of a slave server. YARN [29] allocates a resource container encapsulating a certain number of CPU cores and a certain size of memory and disks for each application.

With the presence of heterogeneous hardware resources in an ADOM cluster, these coarse-grained metrics cannot accurately reflect the computational power of the resources. For example, the family of Intel Core i5 has several modes (e.g., Core i5-750, Core i5-760, and Core i5-750S) with different frequencies (e.g., 2.67 GHz, 2.8 GHz, and 2.4 GHz). The computational power of the CPU core of i5-760 is significantly different from that of i5-750S. Scheduling tasks based on only the number of CPU cores and the size of memory obviously cannot optimize the performance.

We therefore propose a metric called CP (computational power) that can reflect the fine-grained heterogeneity of hardware resources, as follows:

$$CP_i(x, y) = (a_i + b_i + c_i) \times x + d_i \times y,$$

where CP_i represents the computational power of machine i, and a_i, b_i, c_i, and d_i denote the weights of CPU mode, CPU frequency, last level cache, and memory frequency, respectively. x is given by:

$$x = \frac{x_i}{\sum_{i=1}^{N} x_i}, \tag{1}$$

where x_i denotes the number of cores of the system-on-chip i and N is the total number of system-on-chips in the cluster. y is given by:

$$y = \frac{y_i}{\sum_{i=1}^{N} y_i}, \tag{2}$$

where y_i represents the size of memory of the system-on-chip i. The value of a_i is specified in Table 1. Note that the value of a_i can be changed if more different CPU modes are involved. The values of b_i, c_i and d_i are determined as shown in Tables 2, 3, and 4, respectively.

Table 1. Determination of a_i

Processor Mode	a_i
Xeon E3	1
Xeon E5	2
Xeon E7	3

Table 2. Determination of b_i

k (GHz)	b_i
$k < 2$	1
$2 \leqslant k < 3$	2
$k \geqslant 3$	3

Table 3. Determination of c_i

k (MB)	c_i
$0 < k < 10$	1
$10 \leqslant k < 20$	2
$20 \leqslant k < 30$	3

Table 4. Determination of d_i

k (Mhz)	d_i
$k \leqslant 2\,000$	1
$2\,000 \leqslant k < 30\,000$	2

Note that it seems counter-intuitive that CP mixes the CPU and memory capabilities together. However, (1) and (2) indicate that x and y are relative metrics. The definition of CP is therefore reasonable.

4 Implementation

4.1 Hardware Implementation

In this paper, we present an initial implementation of the hardware. The communication board is implemented by using RTL 8316E network switch chip, SR8201G network PHY, and three power control chips: TI TPS5430, G5626, and RT8288. The front side and the back side of the board are shown in Fig. 10 and Fig. 11, respectively. We use the S500 system-on-chip produced by Actions (Zhuhai) Technology Co., Ltd. The S500 system-on-chip integrates a Quad-Core Cortex-A9R4 processor.

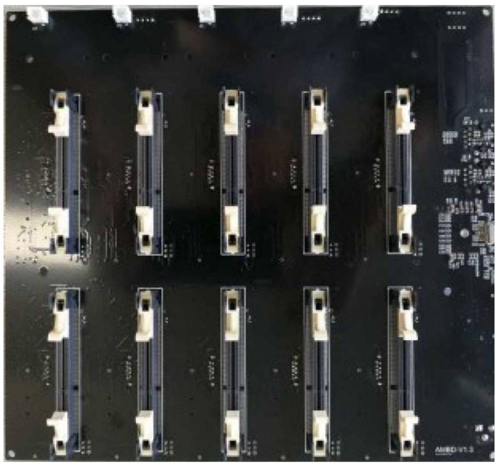


Fig.10. Front side of the communication board.

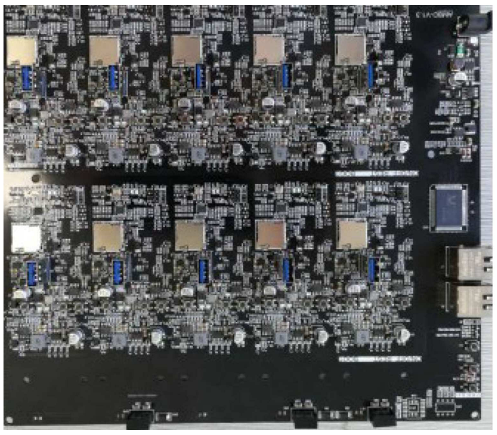


Fig.11. Back side of the communication board.

4.2 Software Implementation

We use C++11 to implement Chameleon with 7 000 lines in total. Moreover, to implement the concurrency programming, we employ the actor model⑤ and use the

⑤https://en.wikipedia.org/wiki/Actor_model, Aug. 2019.

libprocess library[6]. In addition, to make Chameleon easy to extend, we employ several design patterns such as singleton and strategy to implement Chameleon. The source codes of Chameleon can be accessed[7].

Fig.12 shows the important data structures of Chameleon. The behaviors of the actor are similar to the concept of the process in the operating system. One actor could send data to other actors by transmitting messages serialized by Google's Protocol buffers[8] like TasksMessage in Fig. 12. Each actor could have a number of threads to invoke the user-defined functions to handle the received messages. The three planes of Chameleon mentioned in Subsection 3.3.2 are implemented inside the Actor Master. ControlPlane maintains two main classes: Scheduler which links to a scheduling algorithm library and Transformation responsible for switching the two-tier architecture described in Subsection 3.3.1. Similarly, ConfigurationPlane also has a configuration library which contains a number of configuration optimization algorithms designed for different workloads as mentioned in Subsection 3.3.2. Finally, DataPlane contains a Slave-tasks Mapping data structure which maintains the information of the tasks distribution among slaves and the resources allocation of the cluster.

The workflow of job execution of Chameleon is similar to Mesos [28]. The Actor Master provides resources to the upper registered framework by Resourceoffer-Message. Then the framework packages the messages of tasks to the Actor Master which schedules these messages to specified slaves after the processing of its three planes. In the end, the Actor Slave initializes a number of executors to control the runtime of each task.

5 Experimental Setup

We build an ADOM cluster by using six communication boards and correspondingly 60 mobile system-on-chips, as shown in Fig.13. We use 16 nodes of this cluster and one cluster consisting of two Xeon servers as our hardware platform. We co-run several (e.g., 4–12) Spark programs on the cluster and manage the cluster by Chameleon and Mesos respectively.

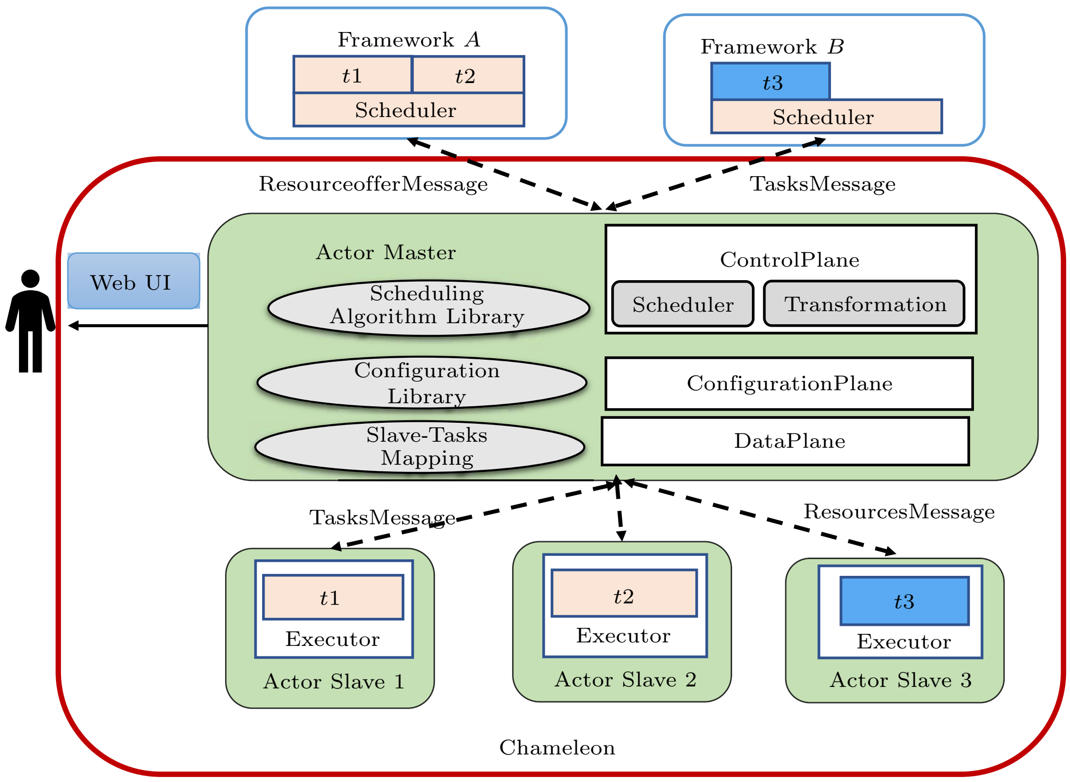


Fig.12. Key data structures of Chameleon. *ti*: task *i*.

[6]https://github.com/3rdparty/libprocess, Aug. 2019.

[7]https://gitee.com/heterogeneous_center/Chameleon, Aug. 2019.

[8]https://github.com/protocolbuffers/protobuf, Aug. 2019.

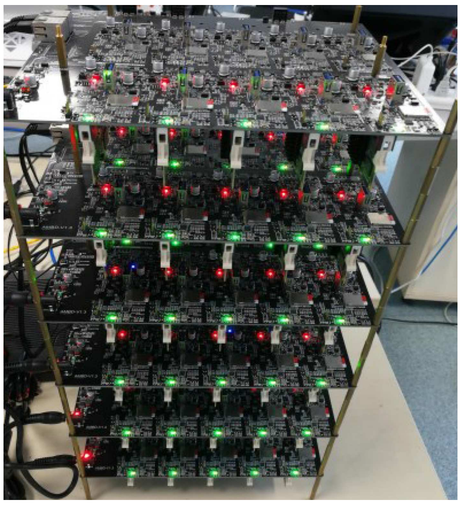


Fig.13. Overview of an ADOM cluster built by six boards.

Table 5 shows the Spark benchmarks used in this study. They represent a sufficiently broad set of typical Spark program behaviors. For example, KMeans has a good instruction locality but a poor data locality while Bayes is the opposite. While both performing selective shuffling, the iteration selectivity of PageRank is much higher compared with KMeans. NWeight is an iterative graph-parallel algorithm implemented by Spark GraphX which computes associations between two vertices that are n-hop away. It consumes a lot of memory in that it stores the whole graph in memory and iterates over the vertices. Finally, WordCount, LogisticRegression, GradientBoostingTree, PrincipalComponentAnalysis, and SingularValueDecomposition are CPU-intensive. TeraSort, Sort, and Join are both CPU-intensive and memory-intensive.

Table 5. Experimental Applications

Application	Abbreviation	Input Data
PageRank	PR	Small/tiny
KMeans	KM	Tiny
Bayes	BA	Tiny
NWeight	NW	Tiny
WordCount	WC	Small/tiny
TeraSort	TS	Tiny
Sort	ST	Tiny
LogisticRegression	LR	Tiny
SingularValueDecomposition	SD	Tiny
GradientBoostingTree	GT	Tiny
Join	JO	Small/tiny
PrincipleComponentAnalysis	PA	Small/tiny

6 Results and Analysis

6.1 Evaluation for the ADOM Cluster

We compare our Chameleon against Mesos by using them to manage the 16-node edge cluster with three cases: co-running four, eight, 12 workloads on the cluster. Fig. 14 shows the average CPU utilization comparison between Chameleon and Mesos when we co-run four workloads (Join, WordCount, PageRank, and PrincipleComponentAnalysis) with their corresponding small input datasets on the cluster. As can be seen, the average CPU utilization of the 16 nodes is significantly lower when they are managed by Chameleon than by Mesos. Since we co-run the same four programs on the same ADOM cluster, this result indicates that Chameleon consumes much less CPU resources than Mesos. In detail, Mesos consumes 4.2x more CPU resources than Chameleon on average and up to 9.4x. The reason is that Chameleon employs the "software-defined" principle, making it only run with necessary functions and consume resources on demand. In contrast, Mesos is not constructed by the "software-defined" principle and it has to run with many unnecessary functions which consume CPU cycles.

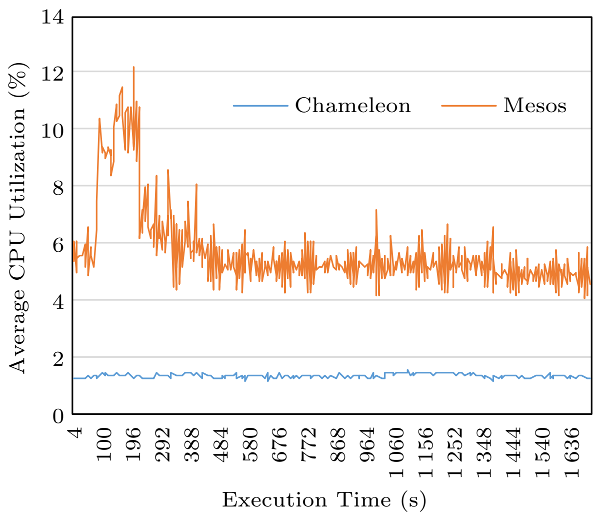


Fig.14. Average CPU utilization when four workloads co-run on the 16-node ADOM cluster over the execution time period.

On the other hand, Fig.15 shows that the four programs consume significantly less memory by Chaneleon than by Mesos. On average, the memory consumption of the programs managed by Mesos is 13.5x of that by Chameleon. This low memory consumption of Chameleon also comes from the "software-defined" design principle which makes Chameleon not load unnecessary functions or libraries into the memory.

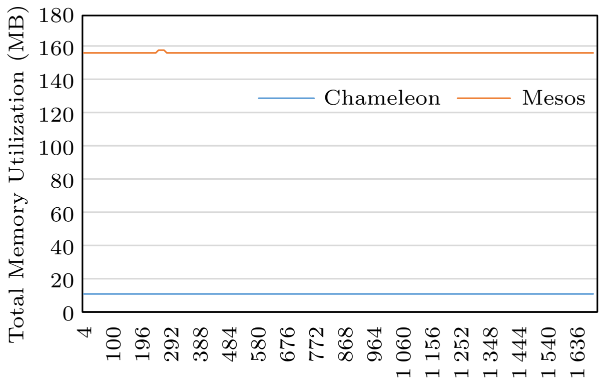


Fig.15. Average memory utilization when four workloads co-run on the 16-node ADOM cluster over the execution time period.

Fig.16 shows the execution time of the four programs when they are managed by Chameleon and Mesos respectively. As can be seen, all the four programs run faster on Chameleon than on Mesos. This is because Chameleon runs with only necessary functions and consumes significantly less CPU and memory resources, and Chameleon can make faster decisions such as task scheduling than Mesos.

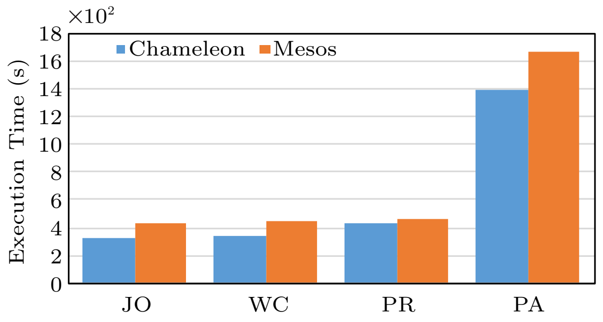


Fig.16. Execution time of four workloads.

In summary, due to the "software-defined" principle employed in Chameleon, it consumes significantly less CPU and memory resources than Mesos. This implies that Chameleon is very suitable for ADOM because the CPU and memory resources are quite limited on ADOM platforms. Moreover, the above results also indicate that we can run more programs on the ADOM cluster with a faster speed. We therefore try to co-run eight workloads and 12 workloads on the 16-node ADOM cluster to observe how Chameleon performs.

Fig.17 shows the average CPU consumption of eight workloads with their corresponding tiny input datasets co-running on the ADOM cluster. As can be seen, the CPU consumption of the eight programs managed by Mesos is on average 3.2x of that by Chameleon. Fig.18 shows that the memory consumption of the eight programs managed by Mesos is 11.2x of that by Chameleon on average.

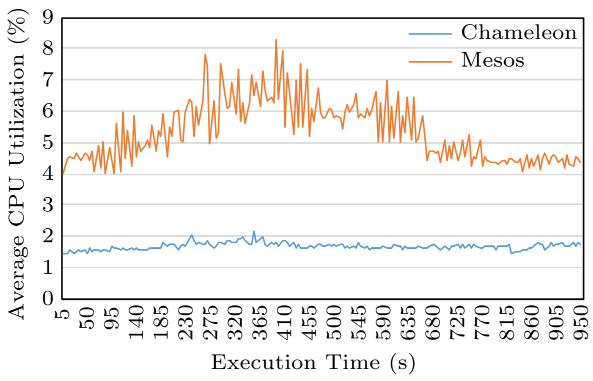


Fig.17. Average CPU utilization when eight workloads co-run on the 16-node ADOM cluster over the execution time period.

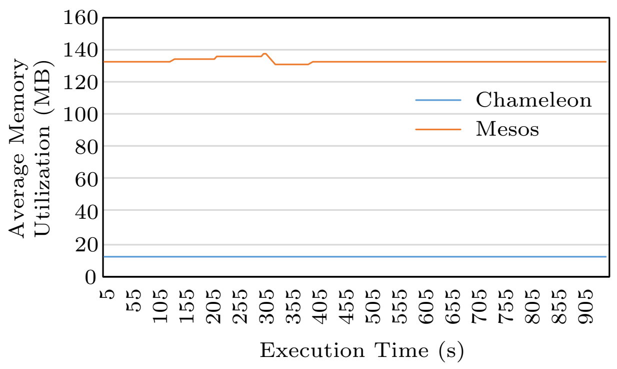


Fig.18. Average memory utilization when eight workloads co-run on the 16-node ADOM cluster over the execution time period.

Fig.19 illustrates that almost all the eight programs run faster on Chameleon than on Mesos. Figs.20 and 21 show that 12 workloads running on Mesos on average consume 2.6x and 11.4x of CPU and memory consumed by these programs running on Chameleon, respectively. Fig.22 shows that the 12 programs still run faster on Chameleon than on Mesos. These results confirm that Chameleon consumes less resources than Mesos but still slightly speeds up programs. This is a good property that is suitable for ADOM.

Fig. 23 shows the throughput for five workloads with their corresponding tiny input datasets when they are managed by Chameleon and Mesos. The average throughput of the five programs by Chameleon is slightly bigger than that by Mesos. However, the improvement of throughput is not distinct and the throughputs of two workloads are even lower when they co-run on Chameleon. This is because Chameleon as a resource manager only does resources allocation and makes faster task scheduling decisions for workloads but does not impact too much on the execution of tasks.

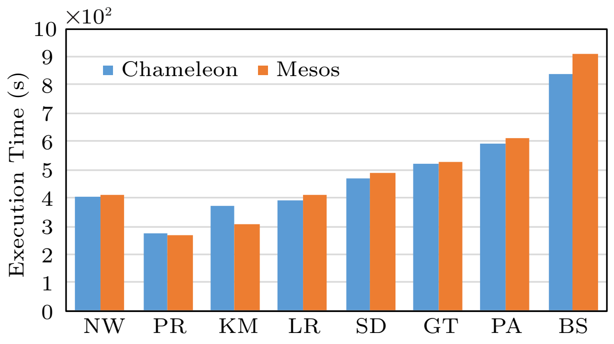


Fig.19. Execution time of eight workloads.

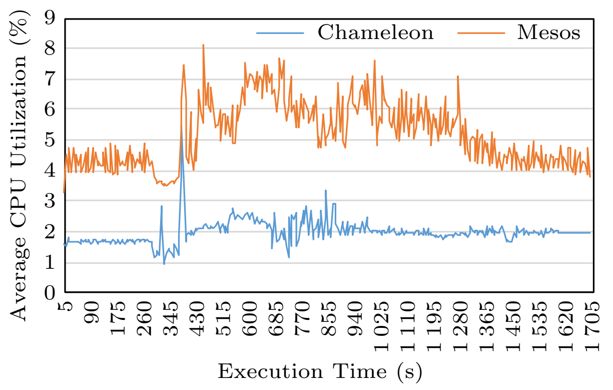


Fig.20. Average CPU utilization when the 12 workloads co-run on the 16-node ADOM cluster over the execution time period.

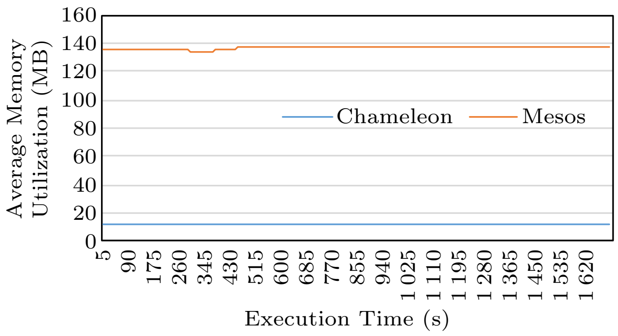


Fig.21. Average memory utilization when the 12 workloads co-run on the 16-node ADOM cluster over the execution time period.

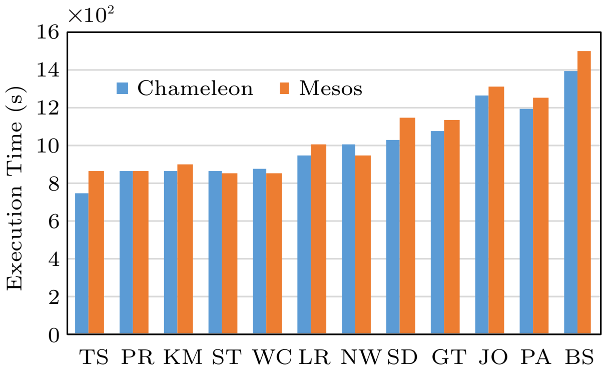


Fig.22. Execution time of 12 workloads.

6.2 Power/Energy Consumption

We now report the power and energy consumption of our SOCA-DOM system. Fig. 24 shows the total power consumption of the 16-node ADOM cluster when running four, eight, and 12 programs respectively. Note that we measure the currency and voltage of the input power lines of the 16-node ADOM cluster, which is an accurate power measurement. As can be seen, the power consumption varies significantly during the program execution for all the three cases. The maximum power consumption can be 56.3 Watt and the minimum is 48 Watt. However, the average power consumptions of the three cases are similar (around 50 Watt), and significantly lower than standard servers.

Fig.25 shows the CPU power consumption when we co-run four and eight programs on two Xeon servers. As can be seen, the power consumption varies much more significantly than that of the ADOM cluster and the highest power consumption achieves 179 Watt. The average power consumptions for the four and eight programs are 61.7 Watt and 66.2 Watt, respectively, which are higher than those on the ADOM cluster.

However, the CPU power consumption is only a small proportion of the total power consumption of a standard server. We therefore compare the energy consumption of the ADOM cluster and the two Xeon servers. Fig.26 shows the results. As can be seen, for the

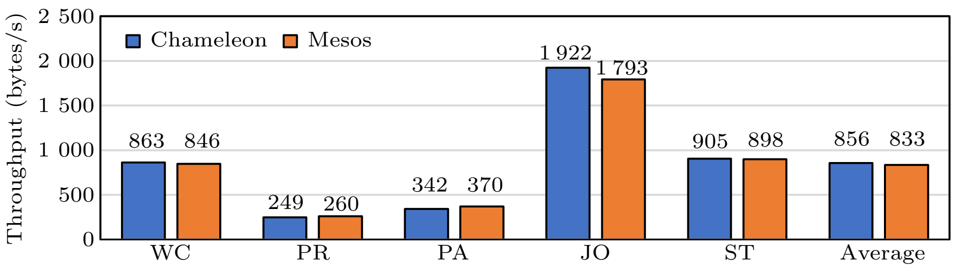


Fig.23. Throughput comparison between Chameleon and Mesos when five workloads co-run on the 16-node ADOM cluster.

same four big data programs, the energy consumption of the Xeon servers is 4x of that of the ADOM cluster. For the eight programs, the Xeon servers consume 2.2x more energy than the ADOM cluster. This indicates that SOCA-DOM is an energy-efficient system.

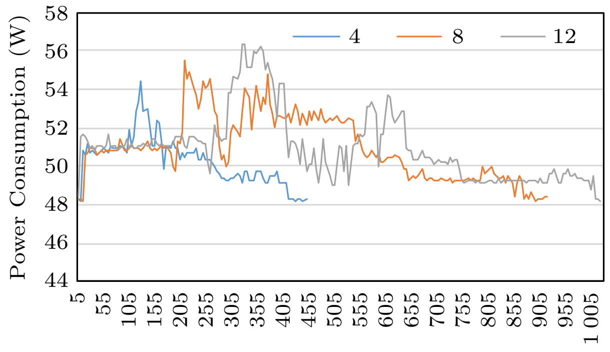


Fig.24. Total power consumption of the 16-node ADOM cluster when running four, eight, and 12 workloads respectively over the execution time period.

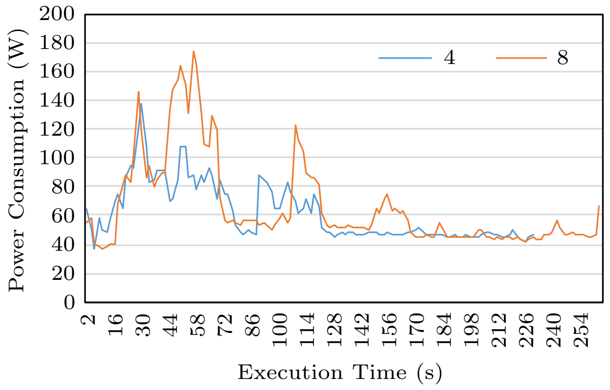


Fig.25. Total power consumption of the CPU of Xeon servers when running four and eight workloads respectively over the execution time period.

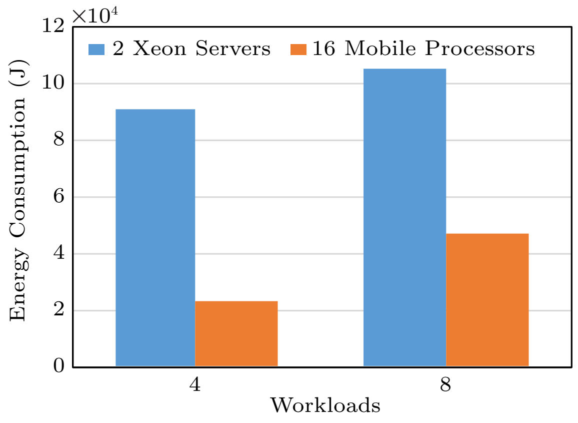


Fig.26. Energy consumption comparison of the 16-node ADOM cluster and 2-node Xeon cluster with four and eight workloads respectively.

6.3 Heterogeneous Cluster Evaluation

As aforementioned, Chameleon can also manage heterogeneous servers. To evaluate the performance of Chameleon in this case, we first use a standard Xeon server and 16 mobile system-on-chips to construct a cluster. We then run the Spark master on the Xeon server and the Spark slaves on the mobile system-on-chips. We also co-run four, eight, 12 workloads on this heterogeneous cluster respectively, but we only report the results for the 12 workloads co-running case. Fig.27 shows the average CPU utilization comparison of the slave nodes which are the mobile system-on-chips. As can be seen, Chameleon consumes significantly less CPU than Mesos in this case. Moreover, the CPU consumption of Chameleon is more stable than that of Mesos. This is also good for the ADOM platform design.

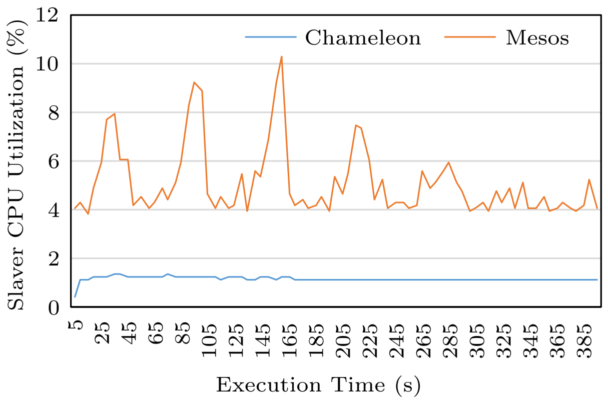


Fig.27. Average CPU utilization comparison when a Xeon server is the master and 16 mobile system-on-chips are slaves over the execution time period of workloads.

However, Fig.28 shows that the master CPU consumption of Chameleon is slightly higher than that of Mesos. The average master CPU consumption of Mesos is 3.8% while that of Chameleon is 4.1%. However, the CPU consumption of Mesos is significantly unstable compared with Chameleon. Nevertheless, achieving slightly higher CPU utilization on standard servers is acceptable since the CPU computational capability of a standard server is powerful.

Fig.29 illustrates the memory consumption comparison between Chameleon and Mesos. As can be seen, Mesos still consumes much more memory than Chameleon in this case. In detail, the size of memory consumed by programs managed by Mesos is 8.7x more than that by Chameleon. Note that we compute the total memory consumption including the master node and all the slave nodes. This indicates that more workloads can also run on this heterogeneous cluster managed by Chameleon.

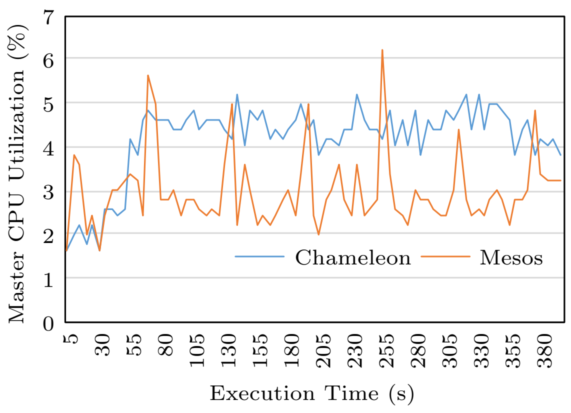


Fig.28. Master CPU utilization comparison when a Xeon server is the master and 16 mobile system-on-chips are slaves over the execution time period of workloads.

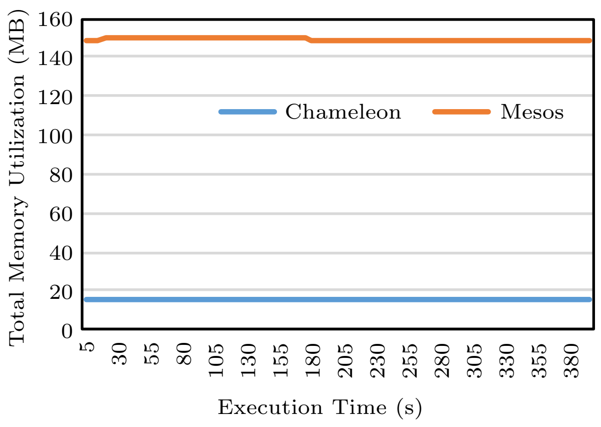


Fig.29. Total memory utilization comparison of the 16-node edge cluster and 2-node Xeon cluster over the execution time period of workloads.

Fig.30 shows the performance of the 12 workloads on the heterogeneous cluster. As can be seen, the programs on Chameleon run significantly faster than on Mesos. On average, programs run 33% faster on Chameleon than on Mesos. For the program WordCount, it runs 4x faster on Chameleon than on Mesos. The reason is that Chameleon designs fine-grained metrics which are used to schedule tasks to execute. The fine-grained metrics can accurately reflect the computational power of different servers. In this case, Chameleon knows that the standard server is much more powerful than the mobile system-on-chips. Consequently, Chameleon can schedule the tasks to proper nodes to improve the performance. In contrast, Mesos employs its fair scheduling policy to allocate resources and leverage the scheduler of Spark to schedule tasks to run, which does not consider the computational power differences between the nodes of a cluster. This is the reason why Chameleon runs much faster than Mesos in the heterogeneous clusters.

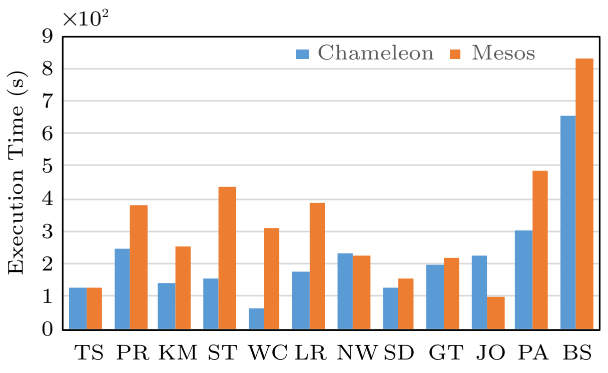


Fig.30. Execution time comparison of the 16-node edge cluster and 2-node Xeon cluster for the 12 workloads listed in Table 5.

7 Limitations

In this section, we discuss the hardware and software limitations of SOCA-DOM.

7.1 Hardware Limitation

As illustrated in Subsection 3.2, each mobile system-on-chip in our hardware design integrates an ARM CPU and an amount of low power DRAM. However, even with the state-of-the-art design techniques, the frequency of our integrated CPU is only 1.8 GHz and the capacity of DRAM is only 4 GB. Since the operating system would take up almost 2 GB to maintain its normal operation, we only have 2 GB left to do big data analytics for tasks per mobile system-on-chip. This results in that SOCA-DOM cannot deal with large-scale datasets (e.g., more than 50 GB) because the data deserialization also requires tens of gigabytes of memory for big data analytics workloads.

7.2 Software Limitation

Our software resource manager Chameleon only provides a number of interfaces for upper application frameworks to do resources allocations. However, the detailed implementations of the interfaces also require users to do a lot of development. In other words, the learning and development cost might be huge for some users who have less knowledge about how Chameleon operates.

8 Related Work

ADOM is a special kind of edge computing which is an emerging computing paradigm and is a hot topic recently. Since there is no study focusing on ADOM, we introduce the related work about mobile edge computing which can be classified by three categories: hardware platform, resource management, and big data analytics of mobile edge computing.

8.1 Hardware Platform

Currently, many researchers proposed to augment the function of the base station servers and made them as the edge computing platform [12, 15, 44]. This may work in some cases but the servers are power hungry and not easy to move, which makes this type of edge computing platforms unsuitable for many more edge computing cases such as IoT (Internet of Things). The closest work to ours is to use Raspberry Pi boards to construct the cloud platform [36]. Although [36] also proposes to use mobile system-on-chips to build computing platforms, it did not propose the resource management approaches like ours and simply used the Ethernet network to connect a number of ARM-based boards, which might occupy a large space when there are many nodes. Another work using wimpy cores is FAWN [45]. [45] proposes to use an array of wimpy cores to build a server. Our work is different from it in the way that we build a communication board to connect the mobile system-on-chips.

8.2 Resource Management

Resource management is a critical topic for computing systems. There are a lot of studies that have been done for resource management including Mesos [28], Yarn [29], Omega [46], Borg [47], Quasar [48], and so on. These are excellent studies aiming at different angles of resource management. However, they were designed only for cloud computing platforms where the resources are rich. In contrast, our work targets for ADOM platforms where we aim to reduce the resource consumption of the hardware resource management software itself.

There are also a large body of studies focusing on task scheduling such as Appollo [49], Tarcil [50], and so on. Our Chameleon differs from them in that Chameleon considers the fine-grained computation differences of CPU cores.

8.3 Applications of Big Data Analysis for Mobile Edge Computing

Because of the data deluge of IoTs, nowadays the number of applications for big data analytics for mobile edge computing is increasing dramatically [1, 10, 13]. Overall, these applications can be classified into batch data-intensive computing and real-time data analytics. For example, the batch data computing framework MapReduce is employed to perform data collection, cleaning, and mining for mobile applications including smart grid [12] and vessel monitoring [26]. Several artificial intelligence algorithms are leveraged to do real-time data analytics on mobile edge platforms such as real-time video analytics with DNNs [23], object detection in self-driving, and so on. Our resource manager Chameleon could provide interfaces of resource allocations for both batch and real-time data analytics applications.

9 Conclusions

In this work, we designed a hardware architecture for analyzing big data on the move by constructing a communication board to integrate an array of mobile system-on-chips. We also proposed a "software-defined" resource manager and developed a prototype named Chameleon. The combination of the hardware and the software is called SOCA-DOM. The experimental results showed that SOCA-DOM consumes up to 9.4x less CPU resources and 13.5x less memory than Mesos. In addition, we showed that a 16-node SOCA-DOM consumes up to 4x less energy than two standard Xeon servers. We concluded that our SOCA-DOM with fine-grained hardware resources and a "software-defined" resource manager works well for analyzing big data on the move.

References

[1] Denby B, Lucia B. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proc. the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2020, pp.939-954. DOI: 10.1145/3373376.3378473.

[2] Sankavaram C, Pattipati B, Pattipati K, Zhang Y, Howell M, Salman M. Data-driven fault diagnosis in a hybrid electric vehicle regenerative braking system. In *Proc. the 2012 IEEE Aerospace Conference*, March 2012. DOI: 10.1109/AERO.2012.6187368.

[3] Xu B, Kumar S A. Big data analysis framework for system health monitoring. In *Proc. the 2015 IEEE International Conference on Edge Computing*, June 27-July 2, 2015, pp.401-408. DOI: 10.1109/BigDataCongress.2015.66.

[4] Jha A K, Nayak S, Veerabhadrappa N K. An architecture for performing real time integrated health monitoring of aircraft systems using avionics big data. In *Proc. the 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution*, Dec. 2017. DOI: 10.1109/CSITSS.2017.8447679.

[5] Wang J, Feng X, Chen Z *et al.* Bandwidth-efficient live video analytics for drones via edge computing. In *Proc. the 2018 IEEE/ACM Symposium on Edge Computing*, Oct. 2018, pp.159-173. DOI: 10.1109/SEC.2018.00019.

[6] Tse P W, Tse Y L. On-road mobile phone based automobile safety system with emphasis on engine health evaluation and expert advice. In *Proc. the 2012 Portland International Conference on Management of Engineering and Technology*, July 29-Aug. 2, 2012, pp.3232-3241.

[7] Nie Y, Zhao J, Liu J, Ran R. Big data enabled vehicle collision detection using linear discriminant analysis. In *Proc. the 10th International Conference on Wireless Communications and Signal Processing*, Oct. 2018. DOI: 10.1109/WCSP.2018.8555647.

[8] Chang W, Chen L, Su K. DeepCrash: A deep learning-based Internet of vehicles system for head-on and single-vehicle accident detection with emergency notification. *IEEE Access*, 2016, 7: 148163-148175. DOI: 10.1109/ACCESS.2019.2946468.

[9] Bonomi F, Milito R, Natarajan P, Zhu J. Fog computing: A platform for Internet of Things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, Bessis N, Dobre C (eds.), Springer, 2016, pp.169-186. DOI: 10.1007/978-3-319-05029-4_7.

[10] Bonomi F, Milito R, Zhu J, Addepalli S, Tan L, Vasudevan V. Fog computing and its role in the Internet of Things. In *Proc. the 1st Edition of the MCC Workshop on Mobile Cloud Computing*, August 2009, pp.13-16. DOI: 10.1145/2342509.2342513.

[11] Mehdipour F, Javadi B, Mahanti A. FOG-engine: Towards big data analytics in the fog. In *Proc. the 14th Int. Conf. Pervasive Intelligence and Computing, 2nd Int. Conf. Big Data Intelligence and Computing and Cyber Science and Technology Congress*, August 2016, pp.640-646. DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.116.

[12] Giang N K, Lea R, Blackstock M, Leung V C M. Fog at the edge: Experiences building an edge computing platform. In *Proc. the 2018 IEEE International Conference on Edge Computing*, July 2018, pp.9-16, DOI: 10.1109/EDGE.2018.00009.

[13] Li Y, Wang S. An energy-aware edge server placement algorithm in mobile edge computing. In *Proc. the 2018 IEEE International Conference on Edge Computing*, July 2018, pp.66-73. DOI: 10.1109/EDGE.2018.00016.

[14] Meurisch C, Seeliger A, Schmidt B, Schweizer I, Kaup F, Mühlhäuser M. Upgrading wireless home routers for enabling large-scale deployment of cloudlets. In *Proc. the 7th International Conference on Mobile Computing, Applications, and Services*, Nov. 2015, pp.12-29. DOI: 10.1007/978-3-319-29003-4_2.

[15] Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 2009, 8(4): 14-23. DOI: 10.1109/MPRV.2009.82.

[16] Reiter A, Prünster B, Zefferer T. Hybrid mobile edge computing: Unleashing the full potential of edge computing in mobile device use cases. In *Proc. the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2017, pp.935-944. DOI: 10.1109/CCGRID.2017.125.

[17] Cartas A, Kocour M, Raman A, Leontiadis I, Luque J, Sastry N, Martinez J N, Perino D, Segura C. A reality check on inference at mobile networks edge. In *Proc. the 2nd International Workshop on Edge Systems, Analytics and Networking*, March 2019, pp.54-59. DOI: 10.1145/3301418.3313946.

[18] Greenberg A, Hamilton J, Maltz D A, Patel P. The cost of a cloud: Research problems in data center networks. *ACM SIGCOMM Comput. Commun. Rev.*, 2009, 39(1): 68-73. DOI: 10.1145/1496091.1496103.

[19] Cuervo E, Balasubramanian A, Cho D, Wolman A, Saroiu S, Chandra R, Bahl P. MAUI: Making smartphones last longer with code offload. In *Proc. the 8th International Conference on Mobile Systems, Applications, and Services*, June 2010, pp.49-62. DOI: 10.1145/1814433.1814441.

[20] Satyanarayanan M, Simoens P, Xiao Y *et al.* Edge analytics in Internet of Things. *IEEE Pervasive Computing*, 2015, 14(2): 24-31. DOI: 10.1109/MPRV.2015.32.

[21] Willis D F, Dasgupta A, WuBanerjee S. ParaDrop: A multi-tenant platform for dynamically installed third party services on home gateways. In *Proc. the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing*, August 2014, pp.43-44. DOI: 10.1145/2645892.2645901.

[22] Kalim F, Noghabi S A, Verma S. To edge or not to edge? In *Proc. the 2017 Symposium on Cloud Computing*, Sept. 2017, pp.629-629. DOI: 10.1145/3127479.3132572.

[23] Liu P, Qi B, Banerjee S. EdgeEye: An edge service framework for real-time intelligent video analytics. In *Proc. the 1st International Workshop on Edge Systems, Analytics and Networking*, June 2018, pp.1-6. DOI: 10.1145/3213344.3213345.

[24] Zhang W, Chen J, Zhang Y, Raychaudhuri D. Towards efficient edge cloud augmentation for virtual reality MMOGs. In *Proc. the 2nd ACM/IEEE Symposium on Edge Computing*, Oct. 2017, Article No. 8. DOI: 10.1145/3132211.3134463.

[25] Wang N, Varghese B, Matthaiou M, Nikolopoulos D S. ENORM: A framework for edge node resource management. *IEEE Transactions on Services Computing*, 2020, 13(6): 1086-1099. DOI: 10.1109/TSC.2017.2753775.

[26] Loghin D, Ramapantulu L, Teo Y M. On understanding time, energy and cost performance of wimpy heterogeneous systems for edge computing. In *Proc. the 2017 IEEE International Conference on Edge Computing*, June 2017, pp.1-8. DOI: 10.1109/IEEE.EDGE.2017.10.

[27] Cappaert J. Building, deploying and operating a cubesat constellation—Exploring the less obvious reasons space is hard. In *Proc. the 32nd Annual AIAA/USU Conference on Small Satellites*, August 2018.

[28] Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph A D, Katz R, Shenker S, Stoica I. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. the 8th USENIX Conference on Networked Systems Design and Implementation*, March 30-April 1, 2011, pp.295-308.

[29] Vavilapalli V K, Murthy A C, Douglas C *et al.* Apache Hadoop YARN: Yet another resource negotiator. In *Proc. the 4th Annual Symposium on Cloud Computing*, Oct. 2013, Article No. 5. DOI: 10.1145/2523616.2523633.

[30] Asanović K. FireBox: A hardware building block for 2020 warehouse-scale computers. In *Proc. the 12th USENIX Conference on File and Storage Technologies*, Feb. 2014.

[31] Lim K, Chang J, Mudge T, Ranganathan P, Reinhardt S K, Wenisch T F. Disaggregated memory for expansion and sharing in blade servers. In *Proc. the 36th Annual International Symposium on Computer Architecture*, June 2009, pp.267-278. DOI: 10.1145/1555754.1555789.

[32] Nitu V, Teabe B, Tchana A, Isci C, Hagimont D. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *Proc. the 13th EuroSys Conference*, April 2018, Article No. 16. DOI: 10.1145/3190508.3190537.

[33] Novakovic S, Daglis A, Bugnion E, Falsafi B, Grot B. Scale-out NUMA. In *Proc. the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2014, pp.3-18. DOI: 10.1145/2541940.2541965.

[34] Klimovic A, Kozyrakis C, Thereska E, John B, Kumar S. Flash storage disaggregation. In *Proc. the 11th European Conference on Computer Systems*, April 2016, Article No. 29. DOI: 10.1145/2901318.2901337.

[35] Klimovic A, Litz H, Kozyrakis C. ReFlex: Remote flash ≈ local flash. In *Proc. the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, April 2017, pp.345-359. DOI: 10.1145/3037697.3037732.

[36] Tso F P, White D R, Jouet S, Singer J, Pezaros D P. The glasgow raspberry Pi Cloud: A scale model for cloud computing infrastructures. In *Proc. the 33rd International Conference on Distributed Computing Systems Workshops*, July 2013, pp.108-112. DOI: 10.1109/ICDCSW.2013.25.

[37] Kreutz D, Ramos F M V, Veríssimo P E *et al.* Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2015, 103(1): 14-76. DOI: 10.1109/JPROC.2014.2371999.

[38] Bei Z, Yu Z, Zhang H *et al.* RFHOC: A random-forest approach to auto-tuning Hadoop's configuration. *IEEE Transactions on Parallel and Distributed Systems*, 2016, 27(5): 1470-1483. DOI: 10.1109/TPDS.2015.2449299.

[39] Yu Z, Bei Z, Qian X. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proc. the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2018, pp.564-577. DOI: 10.1145/3173162.3173187.

[40] Wang Y, Li L, Wu Y *et al.* TPShare: A time-space sharing scheduling abstraction for shared cloud via vertical labels. In *Proc. the 46th ACM/IEEE International Symposium on Computer Architecture*, June 2019, pp.499-512. DOI: 10.1145/3307650.3326634.

[41] Buyya R, Calheiros R N, Son J, Dastjerdi A V, Yoon Y. Software-defined cloud computing: Architectural elements and open challenges. In *Proc. the 2014 International Conference on Advances in Computing, Communications and Informatics*, Sept. 2014, pp.1-12. DOI: 10.1109/ICACCI.2014.6968661.

[42] Jararweh Y, Al-Ayyoub M, Darabseh A, Benkhelifa E, Vouk M, Rindos A. Software-defined cloud: Survey, system and evaluation. *Future Generation Computer Systems*, 2016, 58: 56-74. DOI: 10.1016/j.future.2015.10.015.

[43] Mei H, Guo Y. Toward ubiquitous operating systems: A software-defined perspective. *IEEE Computer*, 2018, 51(1): 50-56. DOI: 10.1109/MC.2018.1151018.

[44] Mor N. Edge computing: Scaling resources within multiple administrative domains. *ACM Queue*, 2018, 16(6): 106-116. DOI: 10.1145/3305263.3313377.

[45] Andersen D G, Franklin J, Kaminsky M, Phanishayee A, Tan L, Vasudevan V. FAWN: A fast array of wimpy nodes. In *Proc. the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, Oct. 2009, pp.1-14. DOI: 10.1145/1629575.1629577.

[46] Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J. Omega: Flexible, scalable schedulers for large compute clusters. In *Proc. the 8th ACM European Conference on Computer Systems*, April 2013, pp.351-364. DOI: 10.1145/2465351.2465386.

[47] Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J. Large-scale cluster management at Google with Borg. In *Proc. the 10th European Conference on Computer Systems*, April 2015, Article No. 18. DOI: 10.1145/2741948.2741964.

[48] Delimitrou C, Kozyrakis C. Quasar: Resource-efficient and QoS-aware cluster management. In *Proc. the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2014, pp.127-144. DOI: 10.1145/2541940.2541941.

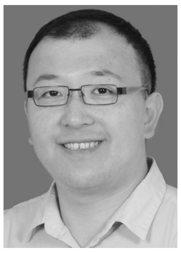
[49] Boutin E, Ekanayake J, Lin W, Shi B, Zhou J, Qian Z, Wu M, Zhou L. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proc. the 11th USENIX Conference on Operating Systems Design and Implementation*, Oct. 2014, pp.285-300.

[50] Delimitrou C, Sanchez D, Kozyrakis C. Tarcil: Reconciling scheduling speed and quality in large shared clusters. In *Proc. the 6th ACM Symposium on Cloud Computing*, August 2015, pp.97-110. DOI: 10.1145/2806777.2806779.

Le-Le Li received his B.S. degree in software engineering from Beihang University, Beijing, in 2016, and his M.S. degree in computer applied technology from University of Chinese Academy of Sciences, Beijing, in 2019. After graduation, he became a research assistant in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, in 2020. Currently, he is a Ph.D. candidate in School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen. His current research interests include cloud computing, edge computing, and big data analytics.

Jiang-Yi Liu received his B.S. degree in software engineering from Northeast University, Shenyang, in 2018, and his M.S. degree in computer applied technology from University of Chinese Academy of Sciences, Beijing, in 2021. His current research interests include cloud computing, power analytics, and big data analytics.

Jian-Ping Fan received his Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences, Beijing, in 1990. He is currently a professor at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen. His research interests include high-performance computing, computer architecture, and architecture supported big data. He is a member of IEEE.



Xue-Hai Qian received his Ph.D. degree in computer science from the Computer Science Department at University of Illinois at Urbana-Champaign, Champaign, in 2013. He is an assistant professor at the Ming Hsieh Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Southern California, Los Angeles. His research interests include graph processing system, storage and cloud systems, and parallelism and concurrency. He is a member of IEEE.

Kai Hwang received his Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1972. He is currently a presidential chair professor in computer science and engineering at The Chinese University of Hong Kong, Shenzhen. His research interests include parallel processing, cloud computing, and network security. He is a fellow of IEEE.

Yeh-Ching Chung received his Ph.D. degree in computer and information science from Syracuse University, Syracuse, in 1992. He is a full professor at The Chinese University of Hong Kong, Shenzhen. His research interests include parallel and distributed processing, cloud computing, and embedded systems. He is a senior member of IEEE.

Zhi-Bin Yu received his Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, in 2008. He is a professor at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen. His research interests include microarchitecture simulation, computer architecture, performance evaluation, and big data processing. He is a member of CCF and IEEE.

JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY

Volume 37, Number 6, November 2022

Content

JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY
《计算机科学技术学报》

Volume 37 Number 6 2022 (Bimonthly, Started in 1986)

Indexed in: **SCIE, Ei, INSPEC, JST, AJ, MR, CA, DBLP**

Edited by:
THE EDITORIAL BOARD OF JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY
Zhi-Wei Xu, Editor-in-Chief, P.O. Box 2704, Beijing 100190, P.R. China
Managing Editor: Feng-Di Shu E-mail: jcst@ict.ac.cn http://jcst.ict.ac.cn Tel.: 86-10-62610746

Copyright ©Institute of Computing Technology, Chinese Academy of Sciences 2022
Sponsored by: Institute of Computing Technology, CAS & China Computer Federation
Supervised by: Chinese Academy of Sciences
Undertaken by: Institute of Computing Technology, CAS
Published by: Science Press, Beijing, China
Printed by: Beijing Baochang Color Printing Co. Ltd
Distributed by:
China: All Local Post Offices
Other Countries: Springer Nature Customer Service Center GmbH, Tiergartenstr. 15, 69121 Heidelberg, Germany
Available Online: https://link.springer.com/journal/11390

ISSN 1000-9000

国内统一刊号: CN11-2296/TP 国内邮发代号: 2-578 RMB ¥160.00