Firework: Big Data Sharing and Processing in Collaborative Edge Environment

Quan Zhang*, Xiaohong Zhang*†, Qingyang Zhang*‡, Weisong Shi* and Hong Zhong‡
*Department of Computer Science, Wayne State University, USA
†School of Computer Science and Technology, Henan Polytechnic University, China
‡School of Computer Science and Technology, Anhui University, China
Email: {quan.zhang, weisong, ge2600}@wayne.edu, xh.zhang@hpu.edu.cn, zhongh@ahu.edu.cn

***Abstract*—Cloud computing, arguably, has become the de facto computing platform for the big data processing by researchers and practitioners for the last decade, and enabled different stakeholders to discover valuable information from large scale data. At the same time, in the decade, we have witnessed the fast growing deployment of billions of sensors and actuators in multiple applications domains, such as transportation, manufacturing, connected/wearable health care, smart city and so on, stimulating the emerging of Edge Computing (a.k.a., fog computing, cloudlet). However, *data*, as the core of both cloud computing and edge computing, is still owned by each stakeholder and rarely shared due to privacy concern and formidable cost of data transportation, which significantly limits Internet of Things (IoT) applications that need data input from multiple stakeholders (e.g., video analytics collects data from cameras owned by police department, transportation department, retailer stores, etc.).**

In this paper, we envision that in the era of IoT the demand of distributed big data sharing and processing applications will dramatically increase since the data producing and consuming are pushed to the edge of the network. Data processing in collaborative edge environment needs to fuse data owned by multiple stakeholders, while keeping the computation within stakeholders' data facilities. To attack this challenge, we propose a new computing paradigm, *Firework*, which is designed for big data processing in collaborative edge environment (CEE). *Firework* fuses geographically distributed data by creating virtual shared data views that are exposed to end users via predefined interfaces by data owners. The interfaces are provided in the form of a set of datasets and a set of functions, where the functions are privacy preserved and bound to the datasets. *Firework* targets to share data while ensuring data privacy and integrity for stakeholders. By pushing the data processing as close as to data sources, *Firework* also aims to avoid data movement from the edge of the network to the cloud and improve the response latency.

I. Introduction

In the big data era, the volume of data is increasing at an unprecedented speed along with the fast development of information technology. The data itself has attracted plenty of attentions of stakeholders due to the underneath valuable information the data contains. Cloud computing has been arguably used as the de facto computing platform for the big data processing by the researchers and practitioners for the last decade. A key promise behind cloud computing is that the data should be already hold in or being transmitted to the cloud and eventually be processed in the cloud. Based on such centralized data processing model, a number of batched [1], [2], [3], [4], [5] and streaming [6], [7], [8], [9], [10], [11], [12] big data processing platforms have been proposed and they suit the cloud services well so far.

At the same time, we are entering the era of Internet of Things (IoT). Billions of sensors and actuators are being deployed worldwide and huge amount of data generated by things are immersed in our daily life. According to the estimation of Cisco [13], [14], 50 billions things will be connected to the network by 2020 and generate 507.5 zettabytes (ZB) data per year due to the increasing machine-to-machine connections, which is 49 times greater than the projected data center traffic (10.4 ZB) for 2019. Given that scale, current cloud computing platform is not economic enough to process all data in a centralized environment in terms of the network bandwidth cost and response latency requirement. Instead, the emerging Edge Computing (a.k.a., fog computing [15], cloudlet [16]) referring to "the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services" [17], is more efficient to process data at the proximity of data sources. Edge computing decentralizes the data storage and performs data processing at the edge of the network (e.g., micro-datacenter [18], cloudlet [16]). To leverage the computation resource in the cloud for IoT applications, recent industrial systems employ message brokers [19], [20], [21], [22], [23] and streaming processing engines to build IoT analytics, including IoT in Google [24], AWS IoT [25], and Quarks [26]. For data processing across geo-distributed data sources, previous studies [27], [28], [29], [30] have extended existing platforms to optimize the network bandwidth usage, data aggregation, query execution, and response latency.

Although cloud computing and edge computing are adopted in most data processing scenarios, an important or fundamental assumption behind them is that data is owned by a single stakeholder, where the user or owner has fully control privileges of the data. As we mentioned, cloud computing requires the data to be preloaded in data centers before a user runs its applications in the cloud [31], while edge computing processes data at the edge of the network but requires closely control of the data producers and consumers. Data owned by multiple stakeholders is rarely shared due to various reasons, such as security concern (e.g., data across border), conflict of interest (e.g., data from competitors), privacy issue (e.g., data of health

978-1-5090-5409-1/16 $31.00 © 2016 IEEE
DOI 10.1109/HotWeb.2016.12

care), and resource limitation (e.g., extremely large and long network distance data transportation) and etcetera.

Taking the cooperation in connected health as an example, the health records of patients hosted by hospitals and customer records owned by insurance companies are highly private to the patients and customers and rarely shared. If an insurance company has the access to its customers' health records, this insurance company could initiate personalized health insurance policies for its customers based on their health records. Another example is "find the lost" in the city [32], where video analytics in crowdsourcing leverages video data from multiple stakeholders across the city. It is common that the police department manually collects video data from surveillance cameras on the streets, retailer shops, individual smart phones, or car video recorders in order to identify a specific lost object, which usually costs a lot of resources and labor hours. If all these data could be shared seamlessly, it can save huge amount of human work and identify an object in real-time fashion. Furthermore, simply replicating data or running analyzing application provided by third party on stakeholders' data may break the privacy and security restricts. Unfortunately, none of the aforementioned can be easily achieved by leveraging cloud computing or edge computing individually.

Our Vision: In this paper, we envision that in the era of IoT, the demand of distributed data sharing and processing applications will dramatically increase because the data producing and consuming are pushed to the edge of the network. To facilitate data sharing and processing in such collaborative edge environment (CEE), we propose a new computing paradigm, ***Firework***[1], that enables distributed data sharing and processing for IoT applications while keeping the data and computation within stakeholders' data facilities. *Firework* fuses geographically distributed data by creating virtual shared data views that are exposed to end users via predefined interfaces by data owners. The interfaces are provided in the form of a set of datasets and a set of functions, in which the functions are privacy preserved and bound to the datasets. By pushing the data processing as close as to data producers, *Firework* aims to avoid data movement from the edge of the network to the cloud and reduce the response latency. A *Firework* instance involves multiple stakeholders and these stakeholders need register their datasets and corresponding functions, which will be abstracted as data views. The registered data views are exposed to all participants in the same *Firework* instance so that any participant can combine multiple data views into a single job to achieve specific data analytics. A job request will be parsed into tasks that are scheduled to corresponding participants' data facilities for execution. More details of the *Firework* framework will be discussed in Section II. *Firework* extends the border of data visibility and provides a new computing paradigm for distributed data sharing and processing in CEE, where resources from the cloud and edge can be employed by *Firework*.

[1]We name the computing paradigm as *Firework* is because the request sent from the customer to the cloud, and the distribution of the many computing tasks from the cloud to the edge are very similar to the procedure of fireworks. Different shapes of fireworks represent the different organization of "shared" data.

The reminder of this paper is organized as follows. Section II introduces the architecture design of *Firework*. In Section III, we use connected health and "find the lost" as case studies to explore potential benefits brought by *Firework*. Then we discuss the opportunities and challenges of *Firework* in Section IV and finally Section V concludes this paper.

II. Firework Computing Paradigm

Firework is designed for data sharing and processing among multiple stakeholders in CEE. As a data producer, a stakeholder can collect data from sensors, cameras, and smart phones, or generated by the stakeholder. As a data consumer, a stakeholder may process not only the data owned by itself (optional) but also data from other stakeholders. A stakeholder also has various local computation capabilities, such as clouds, IoT gateways, and mobile devices. The data is shared through predefined interfaces, which define the accessible datasets and applicable functions upon the datasets. To protect data privacy and avoid large volume data transmission, privacy preserving functions prevent data leakage by sharing sensitive knowledge only to intended users such that one cannot infer the raw data from the outputs, and the size of the functions' output would be at least three orders of magnitude smaller than that of the raw input dataset (e.g., the output of 1TB input data is 1GB). In this section, we describe the preliminary design of *Firework*. In details, we first introduce the terminologies and architecture of *Firework*, then we describe the virtually shared data abstraction, and lastly we compare *Firework* with existing distributed data processing paradigms.

A. Terminology

We introduce the terminologies used in *Firework* to illustrate its major components in this subsection.

- *Distributed Shared Data (DSD)*: It provides a virtual view of shared data in *Firework*. It is worth noting that stakeholders might have different views of DSD.
- *Firework.View*: Inspired by the success of object oriented programming, we define the dataset and bound functions as *Firework.View*, which provides partial/full view of the entire DSD. The dataset describes the data to be shared and the functions define the applicable operations upon the dataset.
- *Firework.Node*: A *Firework* participant/user is denoted as a *Firework.Node*. As a data producer, it publishes datasets, privacy preserving functions, and computation resources, while as a data consumer, it subscribes datasets from other participants. The data pub/sub is carried out by *Firework.View*s, which are available for all participants/users.
- *Firework.Manager*: A *Firework.Manager* is responsible for the management of *Firework.View*s and user jobs. First, a *Firework.Node* registers its *Firework.View* to *Firework.Manager*. Second, the *Firework.Manager* receives, analyzes, and dispatches user jobs to *Firework.Node*s

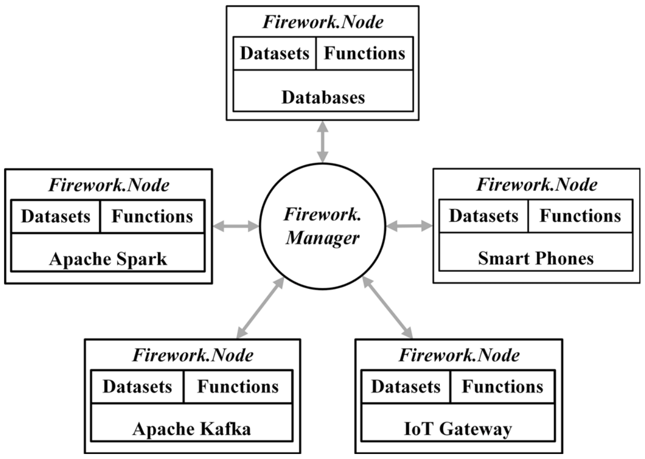

Fig. 1. A high level overview of *Firework*. Each *Firework.Node* could leverage heterogeneous computing platforms and define the dataset and functions.

where the real computation is conducted. The *Firework.Manager* also serves as a job tracker that provides job operation interfaces (e.g., creation, cancellation, and status query). Third, the *Firework.Manager* retrieves intermediate outputs from *Firework.Node*s to facilitate iterative and interactive queries.

- *Firework*: A *Firework* is an operational instance of the *Firework* paradigm. A *Firework* instance consists of multiple *Firework.Node*s and one *Firework.Manager*.

Figure 1 shows a high level overview of a *Firework* instance consisting of five *Firework.Node*s. *Firework* is compatible with all kinds of computing platforms, ranging from clouds with Apache Spark, databases and Apache Kafka to edge devices of smart phones or IoT gateways (e.g., Intel Edison, Raspberry Pi), which allows *Firework* to handle batch and stream data processing in CEE. If all *Firework.Node*s are homogeneous, such an instance will be similar to cloud computing or edge computing. The most important difference between cloud/edge computing and *Firework* is the data ownership, which consequently affects the data movement involving privacy concern and network cost. Another difference is that the size of output data would be much smaller than the input data. The output data here refers to the data returned to the user. There is no restriction for the intermediate data generated by the local computation in a *Firework.Node*, since those data will not be visible to the user. All *Firework.Node*s should define the shared datasets and privacy preserving functions and register them to the *Firework.Manager*. As a major abstraction of DSD, the *Firework.View* is abstracted as a 'class-like' (i.e., the class in C++/JAVA) object for a *Firework.Node*, which describes the private data and public functions. Figure 2 shows a preliminary abstraction for *Firework.View* in JSON format. The 'Datasets' field describes the data to be shared and the 'Functions' field lists all applicable functions for corresponding dataset. This is a conceptual design that does not involve much details of implementation.

```
{
  "FireworkName": "A readable name",
  "FireworkID": "A unique ID of the data owner",
  "ViewName": "A readable name",
  "ViewID": "A unique ID of the view",
  "Dataset": [
    {
      "Name": "A readable name",
      "Description": "A readable description",
      "Metadata": "Available metadata for users",
      "ACL": "Access control list",
      "Schema": "Data set schema"
    }
  ],
  "Functions": [
    {
      "Name": "Function name",
      "FunctionID": "A unique ID specified by data owner",
      "Description": "Functionality description",
      "Input": "Input data",
      "Output": "Output data",
      "Platform": "Programming language, computing model, etc."
    },
    {
      "Name": "Function name",
      "FunctionID": "A unique ID specified by data owner",
      "Description": "Functionality description",
      "Input": "Input data",
      "Output": "Output data",
      "Platform": "Programming language, computing model, etc."
    }
  ]
}
```

Fig. 2. A conceptual abstraction of *Firework.View* in JSON format.

B. Distributed Data Sharing

The major abstraction of the DSD is *Firework.View*, which includes the dataset and functions. Based on the assumption that data is private and no raw data transmission is allowed between *Firework.Node*s, all computation is conducted in data owner's local computing facilities. A virtual data set in *Firework* is an aggregated view of datasets in *Firework.View*s. As an example, a *Firework* instance can define a virtual data set that consists of temperature data citywide, in which real temperature data are the data shared by participants across the city. To share the temperature data, each *Firework.Node* defines its local dataset and functions, which depend on the underlying storage system (where the data is stored, e.g., filesystem, database) and computation framework (how to provide the public functions to others, e.g., REST API, Mapreduce job). The datasets and privacy preserving functions are managed by its owner and only the metadata that a user needed to develop applications are registered at *Firework.Manager*. Once a new dataset and its corresponding functions are defined and implemented, a *Firework.Node* can register it as a *Firework.View*. A data owner can add, remove, and update a *Firework.View*. A user can get all available *Firework.View*s by querying the *Firework.Manager* and choose which *Firework.View*s are involved in his/her computation. The computation should be limited to the functions bound to the dataset and no other operation is allowed.

A user job is represented by a sequence of operations on *Firework.View*s and sent to *Firework.Manager*. Upon the receiving of a user job, a job descriptor is created, where the operation sequence is used to generate a directed acyclic graph (DAG). A job analyzer splits the DAG into different stages

TABLE I
COMPUTING PARADIGM COMPARISON.

Computing Paradigm	**Grid Computing**	**Cloud Computing**	**Service Computing**	**Edge Computing**	***Firework***
Objective	Computing resource share	SaaS/PaaS/IaaS	Service share	Compute at edge	Virtual data share
Functionality Ownership	User specified	User specified	Service provider defined	User specified	Data owner defined
Data Distribution	Local	Geo-distributed	Local	Geo-distributed	Geo-distributed
Raw Data Transfer	Across data centers	Across data centers	Across data centers	Edge to data center	No
Heterogeneity	No	Yes	No	Yes	Yes

and tasks and a job manager dispatches the tasks to involved *Firework.Node*s based on the DAG. When a *Firework.Node* receives a task, it will schedule and execute the task based on its available computing frameworks, resources and schedule policies. Finally, the user fuses the results retrieved from all involved *Firework.Node*s.

C. Computing Paradigm Comparison

To distinguish from other computing paradigms, we compared *Firework* with *Grid Computing, Cloud Computing, Service Computing*, and *Edge Computing* in five dimensions. As illustrated in Table I, *Firework* distinguishes from other paradigms in the following aspects: i), *Firework* provides virtual data sharing among multiple stakeholders while others focus on computation resource sharing; ii), *Firework* allows data owners to define the functions that can be performed on their data. The others collect data from users and define the computation/services by the data facility owners; iii), compared to cloud computing, *Firework* also reduces the data transmission cost by applying functions that generate relatively small size of output data; and iv), *Firework* pushes the computation to the edge of the network, which is similar to edge computing, but *Firework* can leverage the other computing paradigms to act as local data computation facilities, while edge computing usually leverages sensors, mobile devices, gateways, and base stations as computing resources.

Up to this point, we have introduced the new computing paradigm of *Firework*, which distinguishes the most from other computing paradigms in terms of data sharing among multiple stakeholders. In the following section, we will use connected health and “find the lost” as two case studies to show the potential capabilities of *Firework*.

III. CASE STUDY

In this section, we will use connected health and “find the lost” to show how *Firework* facilitates the cooperation among multiple data owners. For the connected health case, it uses a flu outbreak to show how the participants collaborate at the enterprise level. For the “find the lost”, it shows how *Firework* leverages edge devices to find a lost object.

A. Connected Health

In the connected health case, we assume that there are six major participants including hospital, insurance, pharmacy, pharmaceutical, logistics, and government forming a *Firework* instance, as shown in Figure 3. The connected health case

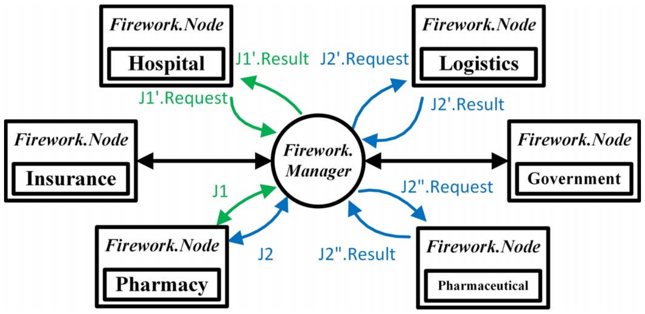

Fig. 3. An example *Firework* instance for connected health. Only the job requests (J1 and J2) of pharmacy are showed.

consists of different types of data owners with distinct computation facilities and each participant provides heterogeneous datasets and functions. In the following flu outbreak scenario, we show how *Firework* brings benefit to the participants..

At the beginning of a flu outbreak, many patients flow to hospitals and the patients’ electronic medical record (EMR) will be updated. A patient theoretically will follow the prescription to get the pills from pharmacies. If the patient did not follow the therapy, the hospital has to take the responsibility for rehospitalization since it cannot get the proof of that the patient did not take the pills. Now with *Firework*, the pharmacy can provide the purchase record of a patient to the hospital, which significantly facilitates the accountable heath care. At the same time, if an increasing flu population is observed by querying the hospitals via *Firework*, which is represented by $J1$ and $J1'$ in figure 3, it will be a wise decision for the pharmacy to provision more flu vaccines. The pharmacy queries pharmaceutical and retrieves the locations, prices and inventories of vaccine warehouses. The pharmacy can also quote for transportation price from the logistics. Then the pharmacy can make a provision plan by solving the cost optimization problem according to the retrieved information. In figure 3, the job $J2$ is split into two tasks $J2'$ and $J2''$, which accomplishes the transportation price query to the logistics and the inventory query to the pharmaceutical, respectively. After the flu outbreak, the insurance company has to pay the treatment bills for the patients. The insurance company can analysis the cost for flu treatment and adjust the policy for the future. Furthermore, the insurance company can also provide personalized health care policy based on a patient’s EMR.

In the connected health case, most participants can benefit from *Firework* in terms of reducing operational cost and

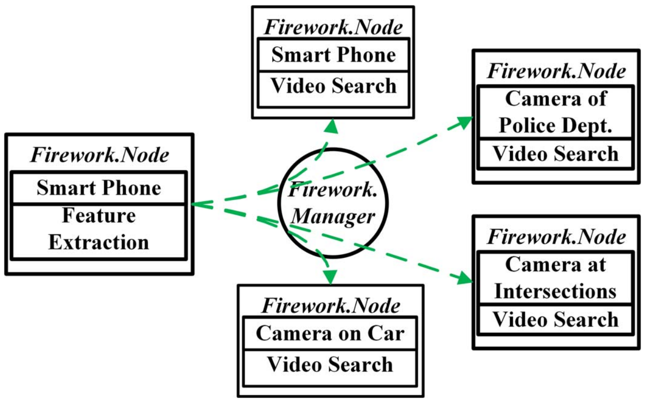

Fig. 4. An example *Firework* instance for "find the lost" in the city. A user sends features of a lost object to other *Firework.Node*s to find the lost object.

improving profitability. However, the hospital in this case could be a pure contributor to health care industry since it is the major data collector in the connected health ecosystem.

B. *Find The Lost*

In the era of IoT, the widespread of cameras are deployed in either fixed locations (e.g., intersection, light pole, store) or mobile carriers (e.g., smartphone, vehicle), which makes video analytics an emerging technology. Cloud computing is not time- and cost-efficient for video analytics since it suffers long response latency and formidable cost of data transportation. Another important concern is privacy because video analytics requires access to data owner's private data, which is unacceptable to send the data to the cloud. In figure 4, we illustrate a *Firework* instance for finding a lost object in the urban area. In such scenario, all *Firework.Node*s provide a video search interface. A user can spread the search request to all connected devices and do the search in real time fashion.

When an object is reported as lost, it is very likely that this object is captured by cameras around at either fixed locations or mobile devices in the urban area. In cloud computing, the video data from the camera has to be uploaded to the cloud to locate the lost object. Even though the data owner pay no attention to privacy, the data transmission is still costly, which makes it extremely difficult and inefficient to leverage the wide area video data. With *Firework* paradigm, a request of searching the lost object can be created at a user's device, where the feature extraction could be conducted. The feature vector then is distributed to all connected devices with cameras and each device performs the object searching using archived local data or real time video stream. The requester collects results from the other devices to locate the lost object. With *Firework*, the video data is no longer necessary to be uploaded to the cloud and the latency is also significantly reduced due to the real time video searching. In this use case, all *Firework.Node*s perform homogeneous video searching task, which is different from the connected health case. An extension of video analytics could be real time object tracking which is common in public security scenarios, where the GPS information of smart phones and vehicles can be used to track a target.

In this section, we used two different types of applications to show that *Firework* can enhance the cooperation between data owners. However, to implement a prototype of *Firework*, a lot of issues need to be addressed in legal and technical level, and in the following section we will introduce several key challenges and opportunities still remaining in *Firework*.

IV. Opportunities and Challenges

Firework provides a novel data sharing and processing paradigm for geographically distributed data-centric applications in CEE. Although *Firework.Node* can directly adopt techniques of cloud computing and edge computing, several issues involving legal and technical problems are still open to discuss, including privacy, programmability, and extensibility, to name a few, and we will bring forward these three challenges and opportunities worth further research.

A. *Privacy*

Data privacy protection is an important issue for any data sharing system. In *Firework*, we assume that data owners define virtual shared data within a *Firework* instance and no extra information can be deduced via exchanging the computing results between any two or more users. Otherwise, the data privacy cannot be guaranteed. One potential solution is to provide privacy preserving functions on edge nodes to prevent data leakage due to backward inferencing, where a privacy preserving function allows the sensitive knowledge to be shared by the intended persons not to everyone to access [33]. This also puts an end to the motivation of obtaining more knowledge via data exchange between participants in different *Firework* instances. However, the privacy preserving function suffers reduced accuracy and increased overhead issues. Furthermore, data sharing among multiple stakeholders requires enforcement authorities to defend the fair competition in both business and market.

B. *Programmability*

In cloud computing, an application is written in one programming language and compiled for a specific target platform, since the program only runs in the cloud. However, in *Firework*, computation is conducted by data owners who most likely have heterogeneous platforms. Although the *Firework.View* defines open datasets and functions, these interfaces could be defined in different ways (i.e., programmed in different programming languages) depending on data owner's computation platform. For a user, it will be difficult to write an application which leverages different interfaces written in various programming languages. This also involves the standardization issue that each *Firework.Node* should provide standard APIs. *Firework* also needs to provide APIs for users to write applications, which can be analyzed by *Firework.Manager* and deployed to *Firework.Node*. Moreover, the collaboration issues (e.g., synchronization, user interaction, etc.) also have to be addressed across multiple layers in *Firework*.

C. Extensibility

As aforementioned, each *Firework.Node* has its own computing facilities. When a data owner joins a *Firework* instance, a bunch of *Firework.View*s could be added. An application may be reprogrammed to adapt the new interfaces provided by the new participant. Similar things could happen when a data owner quits a *Firework* instance. In a more complex case, if one data owner joins multiple *Firework* instances and exposes different sets of datasets and functions, the data privacy may be threatened because a user in one instance can get extra data from another user in the other instance. All these issues require that *Firework* should be able to extend easily so that the transitions could be done seamlessly.

V. Conclusion

In this paper, we have introduced *Firework* as a computing paradigm for big data processing in CEE. *Firework* can leverage data from multiple stakeholders and various heterogeneous computation platforms. We have also introduced the concept of virtual data sharing among stakeholders and illustrated that how emerging data-centric applications take advantage of *Firework* via two use cases (i.e., connected health and "find the lost"). At last, we put forward the challenges and opportunities that are worth working on, and we hope this paper can bring these issues to the attention of the community.

Acknowledgment

This work is supported in part by the Introduction of Innovative R&D team program of Guangdong Province (No. 201001D0104726115), China.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[3] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, and C. Curino, "Apache tez: A unifying framework for modeling and building data processing applications," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1357–1369.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[5] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 423–438.

[7] "Apache storm," https://storm.apache.org/, [Online; accessed April 20, 2016].

[8] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 239–250.

[9] IBM, "Ibm infosphere streams," www.ibm.com/software/products/en/infosphere-streams, [Online; accessed April 20, 2016].

[10] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[11] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.

[12] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: Reliable stream computation in the cloud," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 1–14.

[13] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, pp. 1–11, 2011.

[14] "Cisco global cloud index: Forecast and methodology 20142019 white paper," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf, [Online; accessed April 20, 2016].

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[16] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal (in press)*, June 2016.

[18] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[19] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing." NetDB, 2011.

[20] "Apache flume," https://flume.apache.org/, [Online; accessed April 20, 2016].

[21] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-sa publish/subscribe protocol for wireless sensor networks," in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE, 2008, pp. 791–798.

[22] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. "O'Reilly Media, Inc.", 2013.

[23] "Amazon kinesis," https://aws.amazon.com/kinesis/, [Online; accessed April 20, 2016].

[24] "Google cloud platform: Iot solution," https://cloud.google.com/solutions/iot/, [Online; accessed April 20, 2016].

[25] "Aws iot," https://aws.amazon.com/iot/, [Online; accessed April 20, 2016].

[26] "Apache quarks," http://quarks.incubator.apache.org/, [Online; accessed April 20, 2016].

[27] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 421–434.

[28] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, "Wanalytics: Geo-distributed analytics for a data intensive world," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1087–1092.

[29] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing grouped aggregation in geo-distributed streaming analytics," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 133–144.

[30] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 323–336.

[31] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[32] W. Shi and S. Dustdar, "The promise of edge computing," *IEEE Computer Magazine*, vol. 29, no. 5, pp. 78–81, 2016.

[33] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 439–450.