SoFA: A Spark-oriented Fog Architecture

Neda Maleki*, Mohammad Loni*, Masoud Daneshtalab*, Mauro Conti†, and Hossein Fotouhi*
*School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden
†Department of Mathematics, University of Padua, Padua, Italy
Email: *{neda.maleki, mohammad.loni, masoud.daneshtalab, hossein.fotouhi}@mdh.se, †conti@math.unipd.it

***Abstract*—Fog computing offers a wide range of service levels including low bandwidth usage, low response time, support of heterogeneous applications, and high energy efficiency. Therefore, real-time embedded applications could potentially benefit from Fog infrastructure. However, providing high system utilization is an important challenge of Fog computing especially for processing embedded applications. In addition, although Fog computing extends cloud computing by providing more energy efficiency, it still suffers from remarkable energy consumption, which is a limitation for embedded systems.**

To overcome the above limitations, in this paper, we propose SoFA, a Spark-oriented Fog architecture that leverages Spark functionalities to provide higher system utilization, energy efficiency and scalability. Compared to the common Fog computing platforms where edge devices are only responsible for processing data received from their IoT nodes, SoFA leverages the remaining processing capacity of all other edge devices. To attain this purpose, SoFA provides a distributed processing paradigm by the help of Spark to utilize the whole processing capacity of all the available edge devices leading to increase energy efficiency and system utilization. In other words, SoFA proposes a near-sensor processing solution in which the edge devices act as the Fog nodes. In addition, SoFA provides scalability by taking advantage of Spark functionalities. According to the experimental results, SoFA is a power-efficient and scalable solution desirable for embedded platforms by providing up to 3.1x energy efficiency for the Word-Count benchmark compared to the common Fog processing platform.

***Index Terms*—Fog Computing, Distributed Processing, Spark Programming, IoT, Energy Efficiency.**

I. Introduction

Big Data and Internet-of-Things (IoT) have produced profound impacts to our everyday life by providing smartness in the whole life aspects such as communication and traffic management, health-care, education, and energy management systems [1]. IoT data analytic is performed in a high-throughput cloud environment since large data processing is still a bottleneck of IoT [2]. Although cloud provides extensive processing capacity, still cloud-based solutions are faced with some challenges. Cloud is not always a feasible solution for the expansion of IoT devices and emerging big data applications due to requiring fast response-time and guaranteeing worst-case execution-time. Additionally, End-users are reluctant to risk uploading their data into the cloud for privacy constraints. Finally, cloud computing is a highly energy hungry solution that restricts being a feasible platform for embedded applications. This problem will be more pronounced soon when more than 30% of global datasphere will be generated by embedded IoT nodes by 2025 [3].

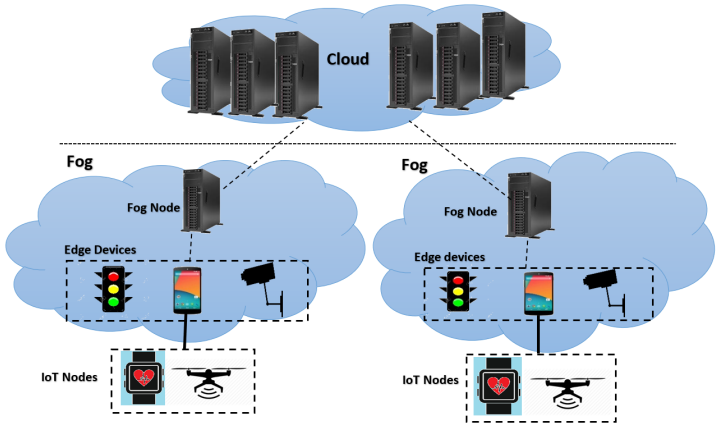

Fig. 1. The Common Fog Architecture.

Fog computing [4], which was proposed by Cisco, is a novel model for analyzing and acting on IoT data. Fog computing makes the business more agile, improves safety, and provides a higher service level. Fog has resolved the cloud computing challenges such as low response-time and low-bandwidth utilization [1]. Some Fog platforms such as Cloudlet, Mobile Cloud, and Ubiquitous have been proposed with different latencies, mobility, costs, and computation capacities [5]. Fig. 1 shows a common Fog architecture with two Fog clusters (one Fog node in each cluster) where the Fog nodes are local middle-throughput machines, compared to high-throughput cloud servers, for processing tasks on the data provided by the edge devices. Section II-A introduces the common Fog architecture with more details.

Fog computing provides new functionalities, while bringing the following challenges. ① The original Fog computing model does not support analyzing large scale data that are produced by IoT applications. ② Its network architecture and service model are not clearly specified [1]. ③ When it comes to data management i.e., data distribution and replication, there is no proper solution in the Fog computing model.

The Fog nodes are expected to analyze and act on the large volume of data generated by thousand of nodes across a large geographic area [4], where continuous generated data are merged into a big stream, which will be sent to the Fog nodes for processing. These data are naturally distributed, therefore, a natural solution to process the data is a distributed stream processing platform.

To tackle the aforementioned limitations, *we introduce*

SoFA, a spark-oriented Fog architecture that leverages spark [6] functionalities to provide scalability, and higher system utilization. SoFA extends Fog functionalities to support distributed job processing on the available edge devices to utilize the whole processing capacity of the system.

According to the common fog architecture (see Fig. 1), the edge devices only work on the data generated by IoT nodes connected to them. However, in the SoFA architecture, the edge devices contribute in the processing of data generated by the other edge devices. Therefore, edge devices can store and replicate the data efficiently, by benefiting from Spark, and run analytic applications, while data is generated at the same time. This paradigm consumes less energy and improves system utilization since if an edge device cannot afford in providing enough processing capacity, SoFA utilizes other edge devices instead of using the local Fog node. We should mention that SoFA is not going to entirely get rid of Fog nodes, but it tries to increase the utilization of system by balancing the processing on the edge devices. In addition, SoFA is only applicable on the edge devices that support Spark platform. Autonomous robots, camera based surveillance systems and health-care T-shirts with embedded sensors connected to a mobile phone -as the edge device- for monitoring body signals are among popular applications that can benefit from SoFA.

Spark is a general-purpose cluster computing system for real-time large-scale streaming data processing with an optimized engine supporting Map-Reduce execution model. When it comes to data management and real-time analytic, the cluster computing play an important role. By using Spark, data could be efficiently managed and replicated across the cluster. Furthermore, some critical security issues such as authentication and access control have been handled in these frameworks through built-in Kerberos [7] framework.

Contribution. In a nutshell, our main contributions are summarized below:

- Analyzing the limitations and requirements of current Fog architectures to support future IoT applications.
- Proposing a distributed Fog platform, called SoFA, that utilizes the available edge devices to both generate and process data. Therefore, common processing on the Fog nodes will be replaced by distributed processing on the edge devices.
- Implementing SoFA on a commodity embedded device over the benchmarks with/without data parallel patterns to evaluate the proposed architecture.
- According to the implementation results, SoFA provides 3.1x more energy efficiency in comparison to the traditional Fog architecture with one Fog node and three edge devices on Word-Count benchmark (See Section 4).

Paper Organization. This paper is organized as follows. In Section 2, we introduce the Apache Spark and its main functionalities used in the SoFA architecture. Section 3 represents the related work in this scope. We delineate the proposed Fog computing architecture and its characteristics in Section 4, and substantiate our argument in favor of Spark as the appropriate distributed system for IoT applications. In Section 5, we show our experimental results in terms of scalability and energy efficiency. Finally, we conclude with how mature would be a Spark-based Fog computing and discussion of future work.

II. Background

In this section, we first introduce the definition of the common Fog architecture used in this paper. Then, Spark processing platform and its major modules will be presented.

A. Common Fog Architecture

Fog computing is considered as an extension of cloud computing which provides the cloud services to the edge of the network with lower latency [8]. In addition, all the cloud computing benefits are preserved in the Fog, including resource efficiency, orchestration, manageability, and virtualization. According to [9], we define the Fog computing in this paper as a model to complement the cloud for decentralizing the concentration of computing resources in data centers towards users for improving the quality of service. According to this definition, our Fog is a three-tier model of cloud resources including Fog nodes,and edge devices. Fig. 1 illustrates the common Fog architecture which contains:

- **Cloud**: Cloud is the remote high-performance computing unit.
- **Fog Nodes**: In general, Fog node could be any device with a processing capability located in the local area network, excluding the IoT devices that are generating the data. In other words, the highest layer in the Fog hierarchy is the Fog node (micro data center), with more computing capacity because it should provide capacity for a larger set of users downwards the hierarchy.
- **Edge Devices**: The processing nodes immediately after IoT nodes in the network which usually have smaller processing capacity compared to Fog nodes. Edge devices are responsible for data processing of their own IoT nodes (mobile smart devices, low-power embedded devices installed on robots, and so on). Edge devices are usually resource-limited (in energy consumption and processing capabilities) embedded devices.
- **IoT Nodes**: The lowest in the hierarchy is IoT nodes which are closer to the edge devices. IoT nodes do not provide any processing capability and only generate data (e.g., end-user devices, IoT sensors and actuators, and so on).

B. Spark Processing Platform

Among all the computing platforms, Spark is a prevalent distributed computing engine since it performs the best in comparison to its competitors; i.e. Hadoop, GraphLab and Flink [1], [10]–[12] in a very large scale distributed environments and for processing real-time streaming data. In addition, Spark has been evaluated and mainly used in large centrally controlled computer clusters [1]. In the following sections, we introduce Spark and its capabilities and then reasons which motivated us to prepare this study.

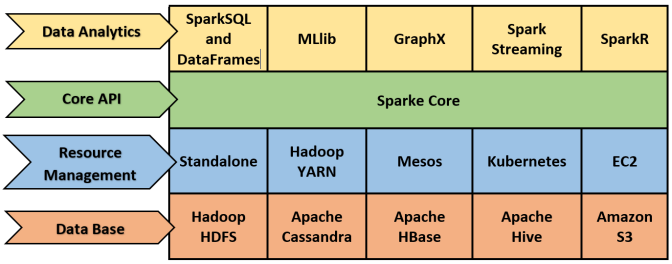

Fig. 2. The Spark Software Stack.

Apache Spark is a fast and general-purpose framework for real-time large-scale data processing in a distributed environment. To accelerate the operations, Spark exploits the in-memory computations [6]. Two key components of Spark are Resilient Distributed Dataset (RDD) and cluster manager [13]. RDD is programming abstraction of Spark which is an immutable and fault-tolerant distributed collection of objects across the cluster nodes and can be operated in parallel [6]. Cluster manager distributes the code, controls the management, distribution and interaction with RDDs, and manages fault-tolerant execution [13]. Spark architecture comprises of a Driver as a Master node and many Executors as Worker nodes. Spark stack consists of the following components that are depicted in Fig. 2 [6]:

- **Spark core** is responsible for memory management, fault recovery, scheduling, jobs distribution and monitoring across the cluster, and interacting with storage systems.
- **Spark SQL** integrates relational processing with functional programming APIs of Spark. It supports querying data either via SQL or Hive Query Language (HiveQL).
- **Hive Query Language (HiveQL)** is an open-source data warehousing method that supports queries expressed in a SQL. Hive is compiled into map-reduce jobs executed on Hadoop.
- **Spark Streaming** is used to process real-time streaming data. It enables high-throughput and fault-tolerant processing of data streaming.
- **MLlib** is Spark API to perform machine learning algorithms.
- **GraphX** is used for graph processing.
- **SparkR** is an R package that provides a light-weight front-end to use Spark from R.

Spark can be deployed in a standalone mode i.e., it uses standalone resource manager. It can also benefit the Hadoop YARN [14], Apache Mesos [15], or Kubernetes [16] resource managers. The standalone resource manager does not support fine-grained access control in a way that other resource managers do so, it would be an important consideration in terms of security issues [6]. Spark can run against Kerberos enabled Hadoop clusters and use secure authentication between its processes [17]. Spark fault-tolerance is provided by two high availability approaches; (i) Standby Masters with ZooKeeper and (ii) Single-Node Recovery with Local File System. In the first approach, Masters in the cluster connect to the same ZooKeeper instance and one of them is elected as a leader and the others will remain in standby mode. If the leader dies, another Master will be elected and the previous Master state will be recovered to resume scheduling. Second approach is used to restart the Master if it goes down. In this case, FILESYSTEM mode can take care of it. Spark coherently manages resources of the system in such a way that application level Quality-of-Service (QoS) constraints are met and resource wastage is minimized [18]. In addition, Spark is polyglot i.e., it supports multiple languages.

III. Related Work

In [1], He et al. have proposed a multi-tier Fog architecture and scalable analytics service to mitigate the problem of huge initial Fog infrastructure investment and give prompt response to fast change of circumstances of smart cities. In this model, Fog nodes divide into two groups: (i) opportunistic computing resources, that consist devices such as smart phones, tablets and vehicles, and (ii) dedicated computing resources, consisting of devices such as small base stations, home hot spots, and macro cellular base stations. Authors have defined each Fog as a cluster of computers with a pool of computing resources. They have considered two types of Fogs in opportunistic tier according to the Spark architecture i.e., many Fog Masters to provide reliability and many Fog Workers.

In [5], authors have proposed an architecture of Fog computing in both computing and network aspects. In addition, they have proposed a framework for resource allocation and latency reduction by considering fault-tolerance and privacy with the possible solutions or optimization methods. Finally, they have evaluated the framework under a given application scenario and Genetic Algorithm combined with a Dirichlet distribution sampling approach.

Bonomi et al. [19] proposed a two-layer Fog software architecture. The first layer which is called abstraction layer corresponds to Openstack that virtualizes the heterogeneous resources with cloud structure. The second layer i.e., orchestration layer is responsible for searching, analyzing, planning and running a job, and can be implemented in a big data platform like Hadoop or Spark. For instance, Openstack is deployed upon the ARM core and use Hadoop, Spark or more specific engines can work via APIs. The API design must consider flexibility, latency, sensitivity and heterogeneity. In [20], authors have presented a multi-tier Fog architecture based on software defined network (SDN) paradigm, which enables service migration to deliver videos with adequate QoE for mobile users. They have considered the operational impacts and benefits associated with such cloud-to-multi-tier Fog migration for video distribution with QoE support. Besides, they have performed the evaluation of the service migration to minimize the traffic in the core network.

Naranjo [21] proposed Fog Computing Architecture Network (FOCAN), a Fog-assisted smart city network architecture. FOCAN has a multi-tier structure in which the applications are running on nodes with simultaneously computation, communication and routing with the other nodes via the smart city environment. FOCAN improves energy provisioning and

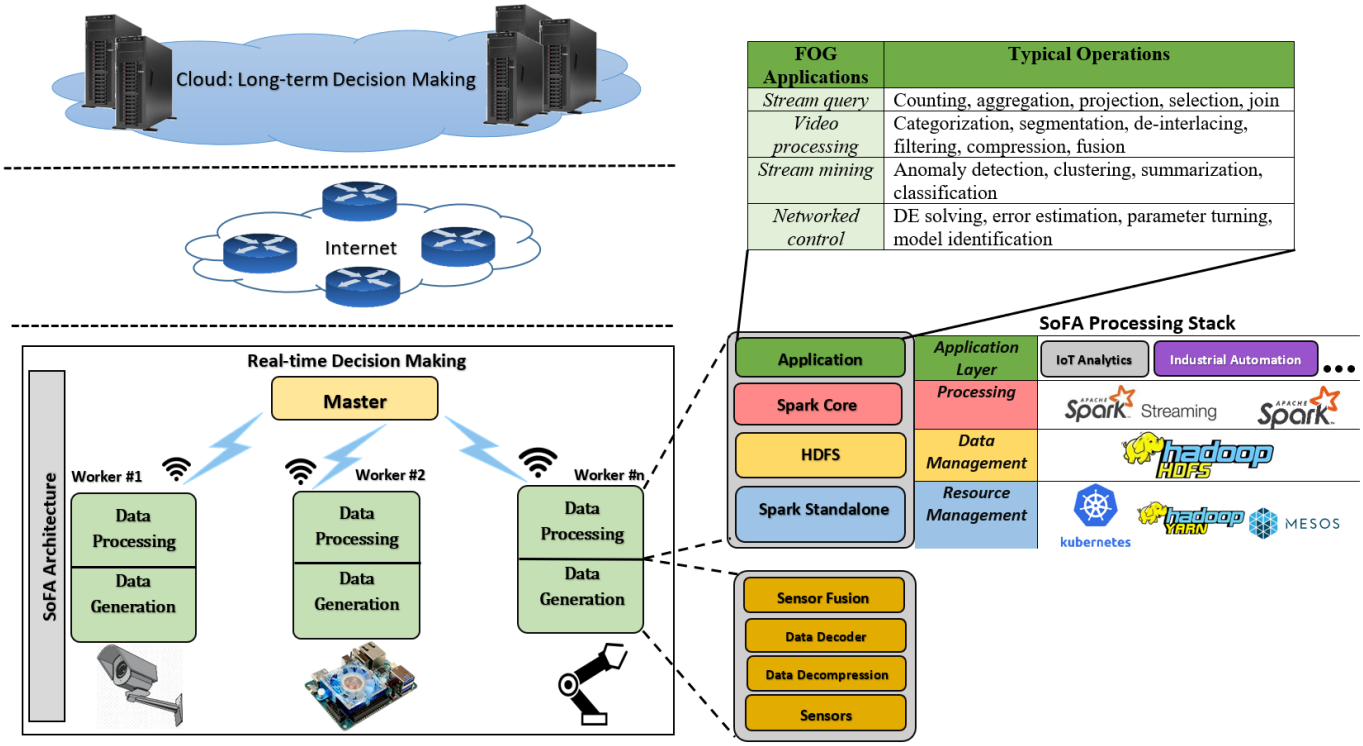

Fig. 3. The Proposed Architecture with SoFA Software Processing Stack installed on the Workers.

the efficiency of provided services while decreasing latency among things with different capabilities. Generally, FOCAN supports three types of communications between devices including interprimary, primary, and secondary communication, for providing required quality of service standards for the Internet of Everything. The main advantages of FOCAN is that the devices can efficiently provide the services with low energy usage.

Yang [22] proposed a Spark-based IoT stream processing for Fog applications. The authors also analyzed the design space of Fog streaming by considering three essential dimensions including human, system, data, and optimization. However, the Fog node is needed since the processing is based on common Fog architecture meaning that Fog nodes act as Spark Master and/or slave nodes .

IV. SoFA Description

We have proposed a Spark-enabled Fog architecture, dubbed as SoFA, in which the edge devices play the role of Fog nodes and the whole Fog functionalities will be embedded into the edge devices. Fig. 3 shows our proposed architecture. In this architecture, edge Master physically co-locates with the other edge Workers and the architecture provides advanced large scale analytic service over edge devices. The overall structure of SoFA and the connections between different modules are illustrated in Fig. 3. The proposed architecture in Fig. 3 contains one Master node known as single tier architecture. SoFA also supports multi-tier architectures by leveraging hierarchical Master topology.

Fig. 4 represents a multi-tier topology where in tier1, non- or less time-sensitive analysis and big data analytic are performed, while tier 2 has been designed to support time-sensitive data processing tasks. As shown in Fig. 4, SoFA has more heterogeneous infrastructure than cloud data centers, although both have similar hierarchical network architecture.

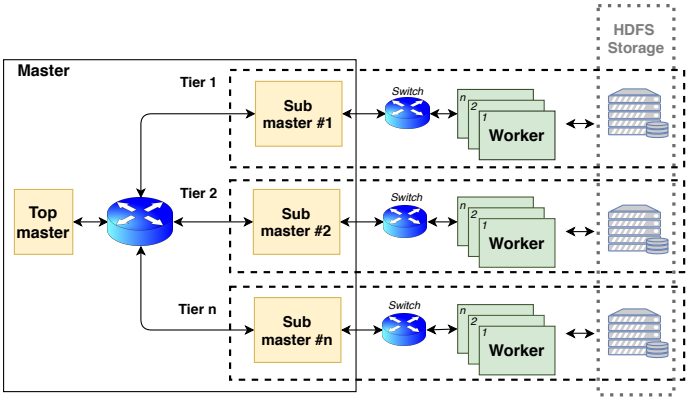

Fig. 4. The Multi-tier Architecture of SoFA Platform.

The key feature of using Spark is utilizing the whole processing capacity by processing data where they are generated to provide a distributed Fog platform. The *Resource Manager* is placed on Master and the *Spark Core*, and *HDFS* are placed on both Master and Workers to store and process the data. The data generated in the edge device will be saved in HDFS file system. Spark manages Workers and dynamic resource

allocation based on the available resources. If one of the Workers are overloaded, Spark migrates the job on a Worker with required data on it.

SoFA processing stack is shown in Fig. 3, which contains four functional layers including (i) Application Layer, (ii) Processing Layer, (iii) Data Management Layer, (iv) Resource Management Layer.

A. Application Layer

This section overviews typical Fog streaming applications [22], shown in Fig. 3.

a) Stream Query and Analytics: A big category of applications for data query and analytics over large data streams can benefit from Fog infrastructure. Gigasight [23] is a typical example of these applications that uses Fog architecture, where it explicitly exploits the Internet edge. Gigasight is a repository of crowd-sourced video streams generated by various cameras where video processing jobs such as video segmentation are carried out locally at a virtual machine (VM)-based cloudlet, and only video metadata is transferred to the cloud for the SQL searches. TinyDB [24] is a database system developed for wireless sensor networks (WSNs) [25]. TinyDB implicitly leverages Fog architecture since the low-power sensor nodes and gateways at the network edge jointly process generated data streams by the sensors.

b) Real-Time Event Monitoring: Event detection applications are based on real-time mining and/or classification of IoT data streams [26]. In these systems, the high-level event detection job contains different low-level mining and/or classification tasks. The processing flow of the event detection job is based on a reversed binary tree topology with the root as the data stream source (i.e., sensors), vertices are miners and/or classifiers, and leaves detect results. These miners and/or classifiers are located in different Fog nodes in a distributed way regarding the available processing resources of these servers.

c) Networked Control Systems for Industrial Automation: The Networked Control System (NCS) is a popular IoT application [27] significantly promoting many critical industrial automation applications. The NCS controlling loop includes controllers, sensors, and actuators that continuously produces real-time data streams including sensor control signals and data flows. Utilizing the fog architecture to process such information provides high-quality communication by minimizing distance between all control units in the Fog architecture. In addition, Fog architecture provides higher-level computing resources that is not supported by the embedded controllers hosted in the limited resource budget embedded devices.

B. Processing Layer

The processing layer is responsible for application-specific processing jobs which is carried out by the Spark core. Recently, many different real-time processing engines have been proposed, such as Flink [12], Apache Storm [28], and Apache Spark Streaming [6]. These stream processing engines are inherently designed for the cloud and/or large-scale data centers, however, they support small cluster of Fog nodes.In this paper, Apache Spark streaming is utilized as the spark core.

C. Data Management Layer

Data management layer is in charge of managing distributed data storage including file systems, data caches, databases, data warehouses, and etc. There has been many data management systems working together with stream processing engines in the cloud, such as Hadoop Distributed File System (HDFS) [29], Apache Cassandra [30] as a NoSQL database, and Apache Kafka [31]. All these data management systems can be applied in the small-scale Fog platforms. By helping the data management layer, edge devices can easily transfer data among each others.

D. Resource Management Layer

This layer mainly focuses on the task scheduling and utilizing system resources, including network and disk I/O bandwidths, processing units (CPUs, GPUs), storage, and managing energy consumption for battery-powered embedded devices [32]. By the help of resource management layer, we can utilize the remain processing capacity of the other edge devices if needed.

All in all, Fog application requirements and the SoFa's response to the Fog requirements are summarized in Table I. SoFA is highly suitable for processing distributed data stream at the edge devices by efficiently managing data replication, job scheduling, and energy consumption.

V. Practical Evaluation

This section presents the evaluation results of SoFA architecture. Job completion time, scalability, and power consumption of the analytics jobs over Spark are recorded as the evaluation metrics. We compared SoFA with a COTS concentrated Fog architecture. The characteristics of the studied Fog architectures in this paper are shown in Table III.

For the evaluations, we have launched a very small setup of Spark cluster with four single-board computers (SBC). The SBCs are connected wireless to a network switch. One of the SBCs acts as a Fog Master, while the rest act as Fog Workers. Spark with the latest version 2.0 is installed in the SBCs, plus we used the standalone Spark manager service in the cluster-mode. Analytics job requests are sent from one of the Fog nodes to the Master node, which dispatches the jobs to the Fog Worker nodes. The nodes are assigned consecutive static IP addresses and the Fog Workers are accessible via pass-wordless SSH from the Master node. In addition, we considered the default block size of the Hadoop configuration equal to 64 MB in all experiments and all the applications has been implemented using Java.

We evaluated the SoFA with three real-world applications including Word-Count, Multilayer Perceptron neural network (MLP), and Pi estimator. Processing streaming data is a key requirement of modern Fog platforms. SoFA is a stream-processing architecture that enables data to be processed when they are generated. In addition, we evaluate the online

TABLE I
SUMMARIZING APPLICATION REQUIREMENTS OF THE FOG PLATFORM AND HOW SOFA ADDRESSES THESE REQUIREMENTS.

Requirement	Detail	SoFA
Scalability	improving the ability of the processing platform by increasing the size of data	SoFA is a in-memory computing platform that uses RDD and its parallel operations to provide a scalable solution
Low Latency	Providing the processing result with low latency after data are generated by the sensors	Removing the data transfer time from edge devices to the Common Fog nodes by local processing of data at the edge Devices (sensor nodes) to decrease latency
High Reliability	Providing a highly reliable system ensure the dependability of the services	Spark is designed to support the loss of any set of Worker nodes. It will rerun any tasks those nodes were executing, and recompute any data it had cached on them.
Energy Efficiency	Modern Fog applications require low-energy consumption especially for embedded mobile applications	SoFA provides a near-sensor processing method which utilizes edge devices to both generate and process data at the same time

TABLE II
THE SPECIFICATION OF STUDIED BENCHMARKS.

Benchmark	Input Size	Description
Word-Count	{50, 500, 900} MB	Counts the occurrence of each word in a text file
PiSpark	# Points: {100, 1000, 10000}	Estimating Pi value (Π) using Monte Carlo
4-Layer MLP	*Arch. #1:* {*4,5,5,3*} *Arch. #2:* {*4,50,50,3*} *Arch. #3:* {*4,500,500,3*}	Multilayer neural network

TABLE III
THE HARDWARE SPECIFICATION USED IN THE IMPLEMENTATION PLATFORM.

Configuration	SoFA: Master/Worker nodes (ODROID-XU4)	Common Fog Configuration
Processor	ARM A15	Intel Core i5-520M
Core/Thread	4/4	2/4
Memory	2 GB	4 GB
Frequency	2 GHz	2933 MHz
Idle Power	4.2	15.1
Max Network B.W.	1000 Mbps	1000 Mbps
Operating System	Ubuntu Mate 16.04	Ubuntu 16.04

processing ability of SoFA by varying the size of processing data. Table II explains the specification of studied benchmarks.

Fig. 5 illustrates the benchmarking framework of SoFA. In the Infrastructure layer, we have the processing resources including wireless-connected SBCs. Spark is used as a distributed processing engine avoiding manual data copy/partitioning to the individual Workers. Moreover, Spark is aware of data distribution and providing efficient data management i.e., we benefit from HDFS for data distribution and replication over the cluster.

ODROID-XU4 is a single-board computer equipped with ARM big-LITTLE architecture [33]. ODROID-XU4 is a cost efficient, low power consumption board while providing good computing power. It has been used for many cost-effective entertainment, surveillance, mobile, and IoT applications. In addition, computing power and storage of SBC ODROID-XU4 have similar features of smart phones and tablets, but has a better user-friendly programming environment. In addition, Hadoop, Spark and Openstack are all announced to be run on ARM processors leading to transplant big data applications to embedded computing platforms [19]. Therefore, SBC is selected in our benchmark as an embedded edge devices.

A. Job Completion Time

Fig. 5 to Fig. 7 represent the total Workers' job execution time for PiSpark, Word-Count, and MLP, respectively. The total Workers initialization overhead is not sensitive to the data input size and on-average is equal to: *PiSpark*= 12 seconds, *Word-Coun*t= 14 seconds, and *MLP*= 21 seconds.

a) PiSpark: According to the Fig. 5, SoFA provides worse execution time by increasing the number of Workers in the PiSpark with 100 estimation points. The reason is that the overhead of the data communication and Workers initializations are more than execution time itself. In other words, SoFA can gain more performance by increasing the number of Workers if we have huge amount of input data sizes. The performance gain of three Workers compared to one Worker with 100 estimation points and 10000 estimation points are equal to **-0.35X** and **+2.0X**, respectively.

b) Word-Count: According to the Fig. 6, SoFA provides less execution time by increasing the number of Workers in the Word-Count application for all different input sizes. SoFA also achieves more performance for big amount of input data sizes by increasing the number of Workers. The performance gain of three Workers compared to one Worker with 50 MB and 900 MB input size are equal to **+2.2X** and **+5.6X**, respectively.

c) MLP: MLP is a machine learning model with intrinsic data dependency. We evaluate the MLP with three different architectures and the same input size to evaluate the impact of SoFA on the applications data dependency. According to the results of Fig. 7, by increasing the number of Workers, for all different architectures, the execution time of SoFA increases. The reason is that Spark benefits from data parallelism for job distribution, while MLP has an intrinsic data dependency and cannot benefit from increasing the number of Workers.

B. Power Efficiency

For measuring energy efficiency, we consider Word-Count as a most representative benchmark with data size of 50MB. We used the kill-a-watt P4400 device to measure the power consumption of Odroid-XU4 board by subtracting the average of idle power from the measured power during benchmark execution (Equations (1)). The power consumption, energy consumption and power efficiency of SoFA is calculated based on the Equations (2) to (4). In Equation (2), α is equal to the number of processing Workers. Equation (3) represents SoFA power consumption which is equal to the addition of Master energy consumption by the Workers energy consumption. $Workers\_Init\_Time$ is the spent time that Master

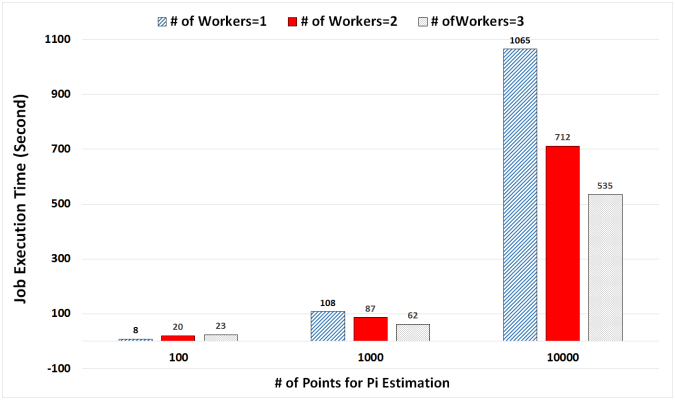

Fig. 5. PiSpark Job Execution Time with Different Sample Sizes on Different Number of Workers.

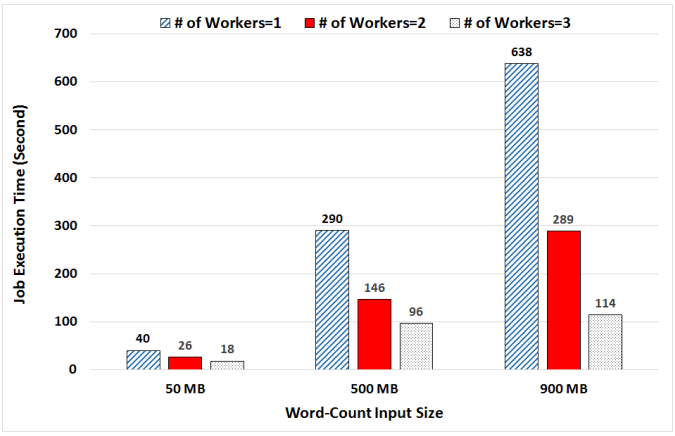

Fig. 6. Word-Count Job Execution Time with Different Input Sizes on Different Nodes.

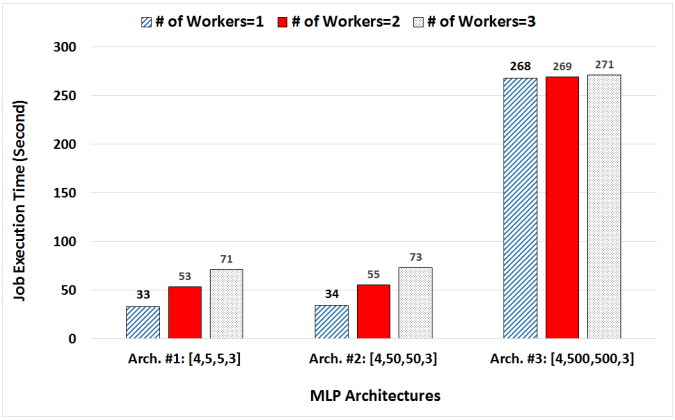

Fig. 7. MLP Job Execution Time with Different MLP Architectures.

TABLE IV
THE CHARACTERISTICS OF EVALUATED PROCESSING PLATFORMS IN SOFA AND THE COMMON FOG ARCHITECTURE.

# Workers	*Parameter*	SoFA: Master/Worker nodes	Common Fog Configuration
1	**Total Completion Time** *	40/15 Sec.	22 Sec.
	Power Consumption †	5.6/2.4 Watt	26 Watt
	Energy Consumption ‡	224/36 J	572 J
2	**Total Completion Time** *	26/15 Sec.	22 Sec.
	Power Consumption †	4/2.5 Watt	26 Watt
	Energy Consumption ‡	104/38 J	572 J
3	**Total Completion Time** *	18/15 Sec.	22 Sec.
	Power Consumption †	2.9/2.7 Watt	26 Watt
	Energy Consumption ‡	52.2/26 J	572 J

* $Workers\_Exec.\_Time$ / $Workers\_Init\_Time$
† $Worker\_Power\_Consumption$ / $Master\_Power\_Consumption$
‡ $Worker\_Energy\_Consumption$ / $Master\_Energy\_Consumption$

initialize the Workers, and $Workers\_Exec.\_Time$ is the total job processing time in Workers. In Equation (4), the energy efficiency of SoFA ($SoFA\_Energy\_Eff.$) is equal to SoFA energy consumption divided by total energy consumption of the common Fog architecture. Table IV compares the energy consumption of SoFA and the common Fog architecture. Fig. 8 presents the comparison of the Energy efficiency of SoFA and the common Fog architecture.

$$\textbf{Power\_Cons.} = AveragePower - IdlePower \quad (1)$$

$$\textbf{SoFA\_Power} = (Master\_Power) + \alpha \times (Worker\_Power) \quad (2)$$

$$\textbf{SoFA\_Energy} = (Master\_Power \times Workers\_Init\_Time) + (\alpha \times Worker\_Power \times Workers\_Exec.\_Time) \quad (3)$$

$$\textbf{SoFA\_Energy\_Eff.} = \left(\frac{Common\_Fog\_Energy}{SoFA\_Energy}\right) \quad (4)$$

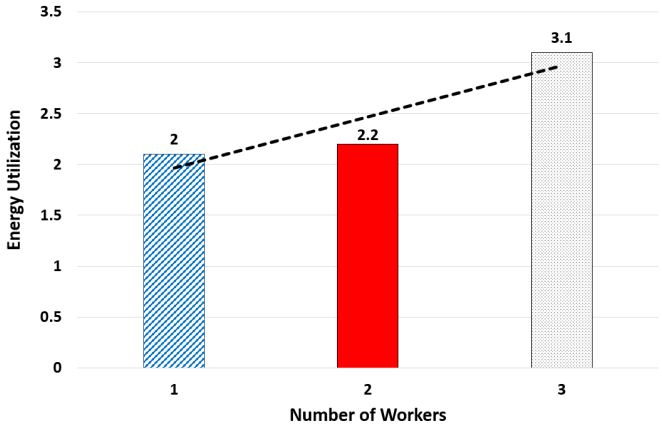

Fig. 8. The Energy Efficiency of SoFA compared to the Common Fog Architecture.

According to the results of Table IV and Fig. 8, although SoFA takes more time to complete the job, it provides more energy efficiency even with one Worker node. In addition, the SoFA job completion time will be decreased by using more Workers leading to gain more energy efficiency up to 3.1x with three Workers compared to the common Fog architecture.

C. Scalability

According to the results of studied benchmarks, SoFA present a scalable solution for the applications without intrinsic data dependency. In other words, total execution time and the energy consumption of Workers are decreased by adding more Workers to the system. The achieved performance and energy efficiency are more significant by increasing the amount of input data.

VI. Conclusion

We envision an Spark-enabled Fog architecture, named SoFA, to be a unified platform for accelerating the IoT applications by using a proper data replication, scheduling and distribution. SoFA is able to run both data distribution and job processing on the edge devices. SoFA distributes jobs among available edge devices to benefit from the low processing capacity of the available edge devices. Therefore, SoFA provides a highly energy efficient solution by maximizing system efficiency suitable for embedded IoT applications. According to the results, SoFA provides up to 3.1x energy efficiency for the Network-intensive and IO-intensive applications like Wordcount compared to the common processing platform.

In the future, we intend to make more experiments, with more number of workers and various types of applications for obtaining a mathematical model to analyze the correlation of performance and scalability in SoFA platform.

VII. Acknowledgement

The authors would like to thank Leo Hatvani and Hamid Reza Faragardi for their helpful discussions and key tips to realize the work. This work was supported by the Swedish Foundation for Strategic Research via the FiC project, and by the Swedish Research Council (Vetenskapsrådet), through the MobiFog starting grant, and by the Swedish Knowledge Foundation (KKS) through the DeepMaker and FlexiHealth Prospekt, and the EU Celtic_Plus/Vinnova project, Health5G (Future eHealth powered by5G).

References

[1] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities," *IEEE Internet of Things Journal*, 2018.

[2] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2013.

[3] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: The evolution of data to life-critical," *Dont Focus on Big Data*, 2017.

[4] Cisco Systems, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," *Www.Cisco.Com*, 2016.

[5] Y. Liu, J. E. Fieldsend, and G. Min, "A framework of fog computing: Architecture, challenges, and optimization," *IEEE Access*, 2017.

[6] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, and S. Venkataraman, "Apache Spark," *Communications of the ACM*, 2016.

[7] J. Garman, *Kerberos: The Definitive Guide: The Definitive Guide*. " O'Reilly Media, Inc.", 2003.

[8] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. da Silva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, 2018.

[9] H. R. Faragardi, "Optimizing timing-critical cloud resources in a smart factory," Ph.D. dissertation, Mälardalen University, 2018.

[10] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[12] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[13] H. Karau, *Fast data processing with Spark*. Packt Publishing Ltd, 2013.

[14] V. Kumar Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O 'malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," *SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing*, 2013.

[15] D. Kakadia, *Apache Mesos Essentials*. Packt Publishing Ltd, 2015.

[16] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure*. " O'Reilly Media, Inc.", 2017.

[17] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[18] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[19] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big data and internet of things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[20] D. Rosário, M. Schimuneck, J. Camargo, J. Nobre, C. Both, J. Rochol, and M. Gerla, "Service migration from cloud to multi-tier fog nodes for multimedia dissemination with qoe support," *Sensors*, vol. 18, no. 2, p. 329, 2018.

[21] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya, "Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments," *Journal of Parallel and Distributed Computing*, 2018.

[22] S. Yang, "Iot stream processing and analytics in the fog," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 21–27, 2017.

[23] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[24] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Transactions on database systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[25] O. Diallo, J. J. Rodrigues, M. Sene, and J. Lloret, "Distributed database management techniques for wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 604–620, 2013.

[26] L. Canzian and M. Van Der Schaar, "Real-time stream mining: online knowledge extraction using classifier networks." *IEEE Network*, vol. 29, no. 5, pp. 10–16, 2015.

[27] R. A. Gupta and M.-Y. Chow, "Networked control system: Overview and research trends," *IEEE transactions on industrial electronics*, vol. 57, no. 7, pp. 2527–2535, 2009.

[28] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[29] K. Shvachko, H. Kuang, S. Radia, R. Chansler *et al.*, "The hadoop distributed file system." in *MSST*, vol. 10, 2010, pp. 1–10.

[30] A. Cassandra, "Apache cassandra," *Website. Available online at http://planetcassandra. org/what-is-apache-cassandra*, p. 13, 2014.

[31] N. Garg, *Apache Kafka*. Packt Publishing Ltd, 2013.

[32] S. Yang, Y. Tahir, P.-y. Chen, A. Marshall, and J. McCann, "Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[33] J. Ivković, A. Veljović, B. Ranelović, and V. Veljović, "Odroid-xu4 as a desktop pc and microcontroller development boards alternative," in *Proc. 6th Int. Conf.(TIO)*, 2016.