J Inf Process Syst, Vol.14, No.4, pp.989~1009, August 2018
https://doi.org/10.3745/JIPS.01.0031

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

An Efficient Implementation of Mobile Raspberry Pi Hadoop Clusters for Robust and Augmented Computing Performance

Kathiravan Srinivasan*, Chuan-Yu Chang**, Chao-Hsi Huang***, Min-Hao Chang***,
Anant Sharma****, and Avinash Ankur****

Abstract

Rapid advances in science and technology with exponential development of smart mobile devices, workstations, supercomputers, smart gadgets and network servers has been witnessed over the past few years. The sudden increase in the Internet population and manifold growth in internet speeds has occasioned the generation of an enormous amount of data, now termed 'big data'. Given this scenario, storage of data on local servers or a personal computer is an issue, which can be resolved by utilizing cloud computing. At present, there are several cloud computing service providers available to resolve the big data issues. This paper establishes a framework that builds Hadoop clusters on the new single-board computer (SBC) Mobile Raspberry Pi. Moreover, these clusters offer facilities for storage as well as computing. Besides the fact that the regular data centers require large amounts of energy for operation, they also need cooling equipment and occupy prime real estate. However, this energy consumption scenario and the physical space constraints can be solved by employing a Mobile Raspberry Pi with Hadoop clusters that provides a cost-effective, low-power, high-speed solution along with micro-data center support for big data. Hadoop provides the required modules for the distributed processing of big data by deploying map-reduce programming approaches. In this work, the performance of SBC clusters and a single computer were compared. It can be observed from the experimental data that the SBC clusters exemplify superior performance to a single computer, by around 20%. Furthermore, the cluster processing speed for large volumes of data can be enhanced by escalating the number of SBC nodes. Data storage is accomplished by using a Hadoop Distributed File System (HDFS), which offers more flexibility and greater scalability than a single computer system.

Keywords

Clusters, Hadoop, MapReduce, Mobile Raspberry Pi, Single-board Computer

1. Introduction

In the present times, massive amounts of data get generated from various sources such as social media, science & technology institutions, weather organizations, corporate firms, web services, and so

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Manuscript received November 15, 2017; first revision December 12, 2017; accepted January 29, 2018.
Corresponding Author: Chao-Hsi Huang (chhuang@niu.edu.tw)
* School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, India (kathiravan.srinivasan@vit.ac.in)
** Dept. of Computer Science and Information Engineering, National Yunlin University of Science and Technology, Yunlin, Taiwan (chuanyu @yuntech.edu.tw)
*** Dept. of Computer Science and Information Engineering, National Ilan University, Yilan City, Taiwan (chhuang@niu.edu.tw; spurs20406 @yahoo.com.tw)
**** Dept. of Computer Science and Engineering, The LNM Institute of Information Technology, Jaipur, India (y13uc025@lnmiit.ac.in; y13uc058@lnmiit.ac.in)

Copyright© 2018 KIPS

on. Such a swift upsurge in data volumes in various fields has captured the attention of the business, academic and scientific communities. Consequently, offering tools for storage, handling and information recovery from vast volumes of data is today among the most significant issues in information technology research [1]. To meet the escalating need for data storage, manipulation and information recovery, new data centers are being created. Moreover, the servers in these data centers guzzle a large proportion of energy resources for storing, analyzing and processing such immense quantities of data, also referred to as big data.

Usually, big data denotes an extensive assortment of enormous data volumes that it is hardly possible to handle and manipulate by employing conventional data management tools, due to its sheer size and complexity [2,3]. The term 'big data' was coined in 2005 by Roger Magoulas of O'Reilly media. The specific and inescapable challenges of big data include the fact that the infrastructure required for handling enormous data volumes has to be built using limited resources, and severely limited processing time-periods. Also, extracting eloquent facts and figures from such data requires the utilization of storage clusters and intricate data applications [4].

Furthermore, these applications should possess features or functionalities such as extreme scalability, data distribution, load balancing, fault tolerance, superior availability, manipulation and information retrieval. To resolve these issues, Dean and Ghemawat [5] established the MapReduce model for processing massive volumes of data on large clusters.

Apache Hadoop is an open source software approach employed in the distributed storage, handling, and analysis of significant data volumes [6]. The crux of Apache Hadoop comprises of a storage portion, referred to as Hadoop Distributed File System (HDFS), and a handling and analysis part termed MapReduce. A single-board computer (SBC) is an all-purpose computer, which is constructed on a single circuit board along with the required microprocessor(s), memory, input/output and other functionalities essential for a well-designed computer [7]. The Mobile Raspberry Pi is a less expensive SBC that can be exploited for many applications. The vital contributions of this research are the deployment of the Mobile Raspberry Pi SBC with Hadoop clusters, which provides parallel and distributed processing with augmented and robust performance. Section 2 illustrates the related work. The system design and implementation are elucidated in Section 3, while the results and discussion are elaborated in Section 4. Section 5 presents the conclusion.

2. Related Work

In recent years, the Raspberry Pi has been proclaimed to be one of the most popular single-board computers around the world. Moreover, SBC clusters have been implemented for several business, scientific and academic tasks. Raspberry Pi SBCs have the competitive advantages of being inexpensive with low energy consumption while offering the functionalities similar to an effusively built computer. Besides, many Raspberry Pi clusters have been constructed and implemented for academia and research utilities [8-11]. Abrahamsson et al. [8] constructed a Raspberry Pi cluster comprising of 300 nodes. Each node in this cluster included a Raspberry Pi Model B device and they constructed it as a low-cost and low-power-consumption cluster for green research test bed and mobile data center applications.

Cox et al. [9] built a low energy consuming, handy, relatively cheap and submissively ventilated

cluster for academic purposes, named the Iridis-Pi cluster. The Iridis-Pi cluster includes 64 Raspberry Pi Model B nodes, in which every node is allocated with a 700-MHz Acorn RISC Machine (ARM) processor, a 256-MB random-access memory and a 16 GB secure digital card for local storage utilities. Kiepert [10] constructed a Beowulf cluster by deploying 32 Raspberry Pi Model B devices, a 48-port 10/100 switch, an Arch Linux ARM, and a Message Passing Interface MPICH3. Besides, the Message Passing Interface program that computes the π value using Monte Carlo technique was employed for measuring the performance of this cluster.

Tso and his colleagues constructed the Glasgow Raspberry Pi cloud comprised of 56 nodes. Each node in this cluster was made up of a Raspberry Pi Model B device along with the PiCloud software stack built over Linux containers for system virtualization. The PiCloud imitates each layer of a cloud stack that varies from virtualization of resources to the behavior of the networks. Additionally, this framework offers an ambiance for academic and cloud computing research [11]. Schot [12] constructed a Raspberry Pi cluster as a substitute for various 1U rack servers. Also, his purpose was to build a low-cost, remarkably synchronized, small-energy and less-expensive ventilated cluster with typical rack servers to be used for a big data setting. Kaewkasi and Srisuruk [13] constructed a model with Hadoop clusters for processing big data, which was built over 22 ARM devices. Subsequently, Hadoop's MapReduce was substituted by Spark, and their research was to study the power consumption and the input/output performance of the hardware. Qureshi et al. [14] used the inexpensive Hadoop clusters to analyze the performance of a cloud computing conceptualization for employing several applications in computer vision and robotics. Hajji and Tso [15] established their research by building a cloud model based on Raspberry Pi for investigating and analyzing real-time big data in different ambient scenarios. Morabito [16] built a model to assess the performance of various low-power gadgets for managing container virtualization.

Compared to the existing frameworks, this research focuses on evaluating the performance of the designed system under realistic conditions. The clusters are formed using the Raspberry Pi third generation units combined with Apache Hadoop to efficiently compute the big data generated. As compared to previous methods, the combination of Hadoop and Raspberry Pi 3 has all the qualities that various methods sometimes failed to adapt. The experiment is easily scalable, cheap, secure, and has high fault tolerance. The low cost of the Raspberry Pi clusters and the scalable Hadoop environment can be used for a variety of complex operations in this era of big data.

3. Background and System Description

In this section, the details about the components such as the SBC, Hadoop, and SURF (Speed Up Robust Features) algorithm are presented.

3.1 Single Board Computer

In general, an SBC is a concise and extensive computer assembled on a single circuit board with microprocessor(s), memory, input/output ports and other functional modules necessary for a full purpose computer. In the present scenario, smart devices and gadgets such as Raspberry Pi, cell phone, Arduino, tablet and notebook computers are some of the commonly available single-board computers.

In comparison with desktop personal computers, an SBC is not dependent on expansion slots for peripheral functions. Currently, the jargon SBC is synonymous with an architecture in which the SBC is plugged into a backplane for fabricating a computer bus. Utilization of an SBC lessens the total cost by decreasing the number of circuit boards required and by also eliminating the need for bus driver circuits and connectors.

3.2 Hadoop

Hadoop is an open source software framework built by Apache. It uses a map-reduce programming model to process big data sets and manage distributed storage. They are built using commodity hardware around the assumption that hardware failures are a common occurrence and should be handled by the framework automatically.

HDFS which handles the storage part is the core of Apache Hadoop, and the map reduced programming model handles the processing part. All the files are split into large blocks in the Hadoop model, which are then distributed among multiple nodes of the cluster. The data is then processed parallelly in these nodes using transferred packaged codes. In this method, each node manipulates and processes the data that it has access to locally. Taking advantage of this data locality allows the processing of the dataset to be highly efficient and faster compared to the traditional supercomputer architecture using the parallel file system. In the conventional architecture, the computation and the data distribution are done via high-speed network, whereas in the Hadoop architecture they are accomplished over the same node.

The Hadoop framework [19] consists of multiple modules like YARN, HDFS, and so on. The libraries and the utilities required by the other Hadoop modules are contained in the Hadoop Common Module. The YARN provides us with the resource management platform capable of managing computing resources across the cluster, allowing them to be used efficiently for user application scheduling. A distributed file-system known as HDFS stores the data on the node. Further, this allows a high aggregated bandwidth across the cluster. The Hadoop MapReduce module is the map-reduce programming model for large-scale data processing. The architecture of two different versions of Hadoop can be seen in Fig. 1.

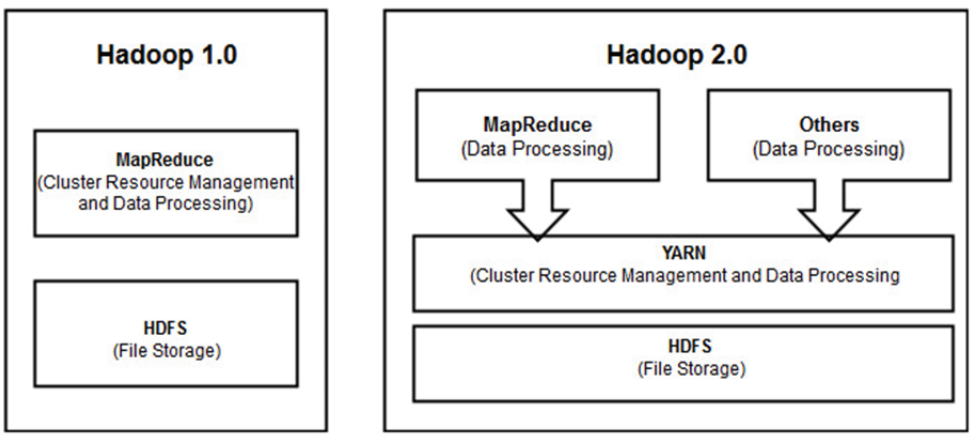


Fig. 1. The architectural difference between Hadoop 1.0 and Hadoop 2.0.

3.3 OpenCV

OpenCV is the Intel developed visual function which provides a large number of functions for image processing. The acronym OpenCV refers to Open Source Computer Vision library, and it has some C and C++ functions that help in implementing standard procedures and algorithms for image processing and computer vision. Some features are based upon state-of-the-art papers, and using them directly instead of reimplementation saves much time. The OpenCV codes are optimized, leading to efficient processing. SURF is the commonly used algorithm implementation of OpenCV to capture image feature points, as explored below.

3.3.1 SURF algorithm

SURF, which is an improvement to the SIFT's functions, was proposed by Bay et al. [17] in 2006. The processing speed is faster compared to the SIFT algorithm, since the computations are significantly reduced. The algorithm is divided into 6 parts.

i) Integral images: The use of integral images is the main reason for the overall performance improvement of the algorithm. Only four addition operators are used to calculate the sum of grayscale values in any rectangle. The Box Filter is used to make the calculations faster.

ii) Hessian matrix based interest points: The roots of Hessian matrix are used to detect the feature points in the SURF algorithm. The detection candidate point is positioned more stably as the determinant is used to determine the local extrema. The Hessian matrix $H(\chi, \sigma)$ is defined in Eq. (4) for an input image Iwith point $\chi = (x, y)$ and the scale σ.

$$H(\chi, \sigma) = \begin{bmatrix} L_{xx}(\chi, \sigma) & L_{xy}(\chi, \sigma) \\ L_{xy}(\chi, \sigma) & L_{yy}(\chi, \sigma) \end{bmatrix} \tag{1}$$

where $L_{xx}(\chi, \sigma)$ a Gaussian function and Box Filter is used to approximate a Gaussian quadratic differential operation. As a result, $D_{xx}(\chi, \sigma)$, $D_{xy}(\chi, \sigma)$, $D_{yy}(\chi, \sigma)$ use DoG faster as compared to the SIFT algorithm.

iii) Scale space representation: The input image is convoluted with the Gaussian function, then reduced and then again convoluted with the Gaussian function. As a result, the SURF algorithm does not change the size of the original image. Further, to achieve different scale changes the appropriate filter size is used, making the algorithm independent of information from the previous layer.

iv) Interest point localization: This part of the algorithm is similar to that of the SIFT, except that the comparative value used here is the determinant of the approximate Hessian matrix. The value is obtained through Eq. (5).

$$\det(H_{approx}) = D_{xx}\, D_{yy} - (0.9\, D_{xy})^2 \tag{2}$$

v) Orientation assignment: The orientation of the point of interest is found to achieve rotational invariance. Moreover, keeping the candidate point as the center, the Haar wavelet responses are found within a circular neighborhood of radius six scale. The Gaussian function then weights the responses. The dominant orientation is estimated by summing the horizontal and vertical responses within a

sliding orientation window of size $\pi/3$. The two summed responses yield the local orientation vector, and the most extended such vector defines the orientation of the point of interest.

vi) Descriptor based on the sum of Haar wavelet responses: A square region of size 20 scale is extracted keeping the point-of-interest as center and orientation as found in the previous steps. This region is further split into 4×4 sized subregions, and the Haar wavelet responses are extracted at 5×5 regularly spaced sample points for each subregion. The descriptor is thus represented as a vector representation of the dimensions.

4. System Design and Implementation

In this section, we introduce the method and the experimental setup used for the research. First, the "Development Tools and System" section explains the reason for the selection, and then the parallel cluster setup is explained in the next two sections. Finally, the "Image Feature Point Processing" section explains how to determine if the image feature points are similar.

4.1 Development Tools and System

Eclipse LUNA is used to develop the Hadoop MapReduce calculation program. MapReduce is a java code written using Eclipse development platform and then exported to a JAR file. This file is then sent to Hadoop run.

4.2 Experimental Equipment and System Platform

We use several sets of Mobile Raspberry Pi to build several Hadoop clusters. For each cluster unit, we can compare the image feature points. First, we create the Hadoop platform to calculate the massive image feature points, and then we observe the effects of these calculations on each cluster. Table 1 below shows the experimental setup used in the process.

Table 1. Details about the experimental setup used in the process

	Cluster						Single computer
	Unit 1	**Unit 2**	**Unit 3**	**Unit 4**	**Unit 5**	**Unit 6**	
Hardware equipment	Mobile Raspberry Pi 3rd generation						I5-4440 8G RAM
Assignment system	Ubuntu						
Cluster system	Hadoop 2.7.1						
Node quantity	5	6	7	8	9	10	1

4.2.1 Setting the cluster network environment

In Hadoop, one master is used to control all the nodes on the DataNode and TaskTracker, as Hadoop relies on SSH for communication with other nodes and data transmission. A LAN setup connects all the nodes, and only the master can communicate outside the network. This design helps avoid exposure of all the nodes on a public network and also provides a faster transmission rate. Table 2 displays an example of the network configuration used for a cluster.

Table 2. Example of network configuration for a cluster

Computer name	Network quality	Task
Master	192.168.1.99 203.145.205.222	NameNode, JobTracker, ResourceManager
Node 1	192.168.1.21	DataNode, TaskTracker, NodeManager
Node 2	192.168.1.22	DataNode, TaskTracker, NodeManager
Node 3	192.168.1.23	DataNode, TaskTracker, NodeManager
Node 4	192.168.1.24	DataNode, TaskTracker, NodeManager
Node 5	192.168.1.25	DataNode, TaskTracker, NodeManager
Node 6	192.168.1.26	DataNode, TaskTracker, NodeManager
Node 7	192.168.1.27	DataNode, TaskTracker, NodeManager
Node 8	192.168.1.28	DataNode, TaskTracker, NodeManager
Node 9	192.168.1.29	DataNode, TaskTracker, NodeManager
Node 10	192.168.1.30	DataNode, TaskTracker, NodeManager

4.3 Hadoop Settings

While establishing the Hadoop cluster units, some of the necessary parameters of the nodes and the master should be the same, while parameters such as memory settings, CPU usage, and so on can be different. Each slave node reports for the resources to the master, which further assigns the unified workload. The main profiles in Hadoop 2.X include Hadoop-env.sh, mapred-env.sh, slaves and yarn-env.sh, which can be adjusted according to individual needs and the hardware. The primary setting includes Java path, Java memory use, run file path, several resource configuration cutting, LOG record file location, LOG memory use, computing resource usage and MapReduce memory usage and slaves number name, and so on. Most of these settings are configured by the master, while slaves only configure a few settings including core-site.xml, HDFS-site.xml, YARN-site.xml and mapred-site.xml, as explained further.

4.3.1 Core-site setting

The primary purpose of this parameter file is to specify the NameNode hostname and the network traffic port number used, which is set to 9,000 for this research. Fig. 2 shows the details regarding the other parameter setting of core-site.xml.

Fig. 2. Code snippet for core-site.xml.

4.3.2 HDFS-site setting

This parameter file controls the HDFS host-node name storage path, data storage path, and so on as shown in Table 3. We set the master to HDFS, master communication port to 50070, and the recovery NameNode communication port to 50090. The file copy number is set to three parts that divides each portion of the block to be copied into three parts. The size of each block is set to 64 MB, which helps prevent wastage of space. The speed of operation is not affected due to the above parameter adjustments. Fig. 3 shows the details regarding the other parameter settings of hdfs-site.xml.

Table 3. Some parameters and their details for hdfs-site.xml

Parameter ID	Remark
dfs.http.address	Set hdfs host computer and port
dfs.secondary.http.address	Set up a backup NameNode
dfs.datanode.data.dir	Set hdfs computer data storage path
dfs.namenode.name.dir	Set the NameNode data storage path
dfs.replication	Set the number of data copies
dfs.block.size	Cutting data set for each part size (default 128MB)

hdfs-site.xml × | mapred-site.xml × | yarn-site.xml × | core-site.xml ×

```
<configuration>
 <property>
     <name>dfs.http.address</name>
     <value>master:50070</value>
 </property>
 <property>
    <name>dfs.secondary.http.address</name>
    <value>chaopi:50090</value>
 </property>
 <property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/home/chao/hdfs/datanode</value>
 </property>
 <property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/home/chao/hdfs/namenode</value>
 </property>
 <property>
  <name>dfs.replication</name>
  <value>3</value>
 </property>
 <property>
   <name>dfs.block.size</name>
   <value>67108864</value>
 </property>
</configuration>
```

Fig. 3. Code snippet for hdfs-site.xml.

4.3.3 Mapred-site setting

This XML parameter file manages the use of some parameters as shown in Table 4 and the number of resources used by map-reduce at the time of execution. Usually, the map and reduce operations do not start at the same time. The reduce operation usually starts after completion of up to 5% of the map operation. Fig. 4 shows the details regarding the other parameter settings of mapred-site.xml.

core-site.xml × hdfs-site.xml × mapred-site.xml × yarn-site.xml ×

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.map.memory.mb</name>
    <value>682</value>
  </property>
<property>
    <name>mapreduce.reduce.memory.mb</name>
    <value>1364</value>
 </property>
<property>
      <name>mapred.reduce.slowstart.completed.maps</name>
      <value>0.98</value>
</property>
</configuration>
```

Fig. 4. Code snippet for mapred-site.xml.

Table 4. Some parameters and their details for mapred-site.xml

Parameter ID	Remark
mapred.reduce.slowstart.completed.maps	Set the start time of reduce, the maximum value is 1 (100%), default value 0.05 (5%).
mapreduce.map.memory.mb	How much memory can be used when setting map execution
mapreduce.reduce.memory.mb	Set how much memory can be used when implementing reduce.

4.3.4 Yarn-site setting

This parameter file controls the adjustment of the usage of the resources as well as the some of the virtual memory, NodeManager (NM) related parameters and so on, as shown in Table 5. The virtual CPU number setting is divided into ResourceManager (RM) with relevant parameters. The communication port of RM and NM is set to 8025 in this experimental setup. The port 8050 is set for the client and RM communication purposes that are used to submit the job and kill the tasks.

Table 5. Some parameters and their details for yarn-site.xml

Parameter ID	Remark
yarn.nodemanager.resource.cpu-vcores	Virtual CPU amount, default 8
yarn.nodemanager.resource.memory-mb	The total amount of memory
yarn.scheduler.minimum-allocation-mb	Minimum memory that can be used by job, default 1 GB
yarn.scheduler.maximum-allocation-mb	Maximum memory that can be used by job, default 8 GB
yarn.nodemanager.vmem-pmem-ratio	Virtual memory using the magnification value, 1 MB physical memory can be applied to the number of virtual memory, the default magnification 2.1

The number of virtual cores used by the entire node is adjusted to the same number with the core of physical value, since the Raspberry core clock is not high. We do not change the lower and the upper

limit values on the memory usage of a single task, which are 1024 MB and 8192 MB, respectively. The other parameter settings can be seen in Fig. 5.

```
<configuration>
<property>
 <name>yarn.resourcemanager.hostname</name>
 <value>master</value>
</property>
<property>
 <name>yarn.nodemanger.local-dirs</name>
 <value>file:/home/chao/hdfs/yarn</value>
</property>
<property>
 <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
 <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
 <name>yarn.nodemanager.aux-services</name>
 <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>master:8025</value>
 </property>
 <property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>master:8030</value>
 </property>
 <property>
  <name>yarn.resourcemanager.address</name>
  <value>master:8050</value>
 </property>
```

```
<property>
  <name>yarn.nodemanager.resource.cpu-vcores</name>
  <value>4</value>
 </property>
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>2046</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.resource.mb</name>
  <value>1364</value>
 </property>
<property>
  <name>yarn.scheduler.minimum-allocation-mb</name>
  <value>682</value>
</property>
<property>
  <name>yarn.scheduler.maximum-allocation-mb</name>
  <value>2046</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.command-opts</name>
  <value>-Xmx1091m</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-pmem-ratio</name>
    <value>4</value>
</property>
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>true</value>
```

Fig. 5. Code snippet for yarn-site.xml.

4.4 Image Feature Point Processing

4.4.1 Feature point extraction

We use SURF algorithm on the input image in OpenCV to extract the feature points of the image. The matrix will consist of N × 64 feature point values, where N is the number of feature points. Each feature point is represented as a 2-dimension array that stores the comparison values at the beginning of each feature point corresponding to the N × 64 matrix and the vector values taken for each feature point, as shown in Fig. 6.

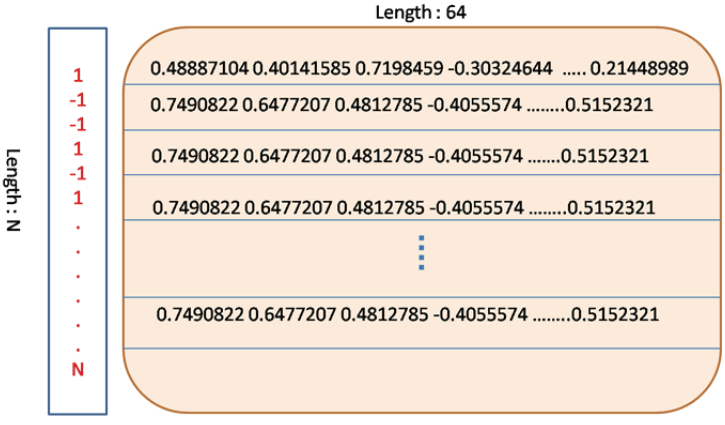
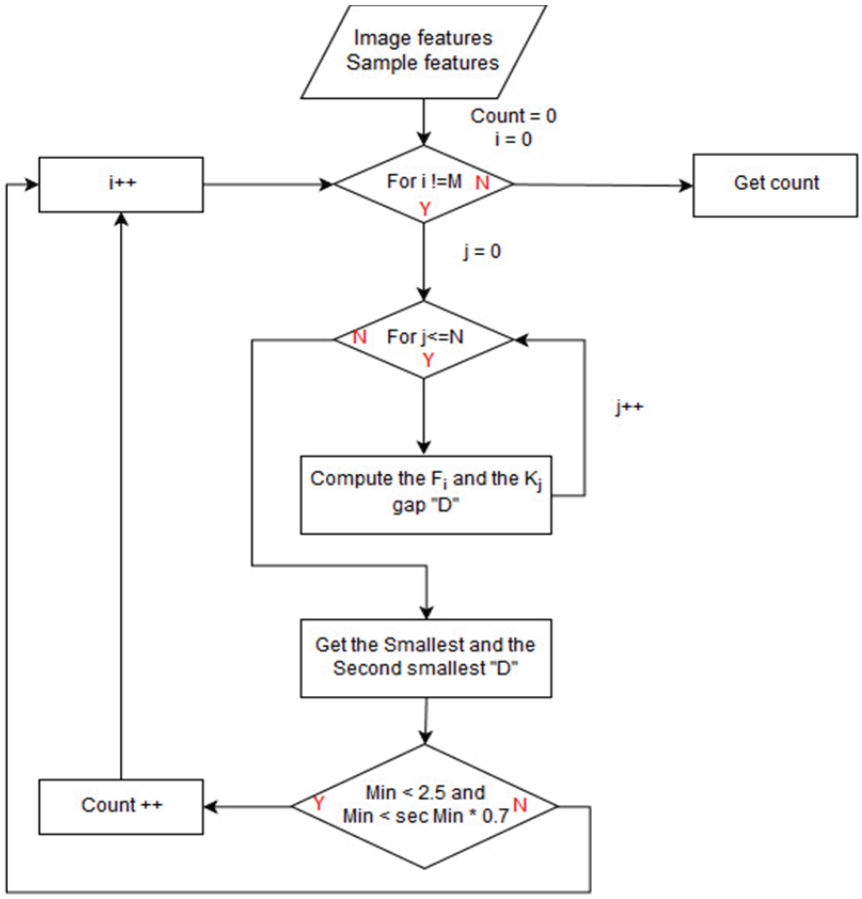


Fig. 6. SURF feature point matrix.

4.4.2 Scale distance algorithm

Each feature point constitutes 16 subregions, and each subregion contains four values. Thus each feature point is a $16 \times 4 = 64$ dimensional vector. We compare the feature points that are similar with regard to scale distance. A threshold value is used to judge the similarity of two feature points. If the feature points are not within the threshold value, we do not compare them, as shown in Fig. 7.

If we do not set the threshold value, there might be a possibility that the algorithm ends up comparing values that are relatively close to the scale but not within the threshold. Typically, the smaller the threshold value, the more stringent the model will be, but if the value is too small it may lead to redundancy, since two images of the same thing will not have the same value of the feature point.



M: The number of feature points in the source
N: Number of feature points of data samples
F_i: Sources sample feature vector
K_j: Data sample feature vector
D: Dimensional spaces distance
Count: Counter

Fig. 7. The feature-point scale algorithm flow chart.

4.5 Performance Evaluation

We first use OpenCV to extract all the image feature points and then store them in a text format with the help of Hadoop HDFS. The Map-Reduce program runs the scale distance algorithm in the map block that counts all the matching samples and the source images at the scale of the feature points. The information regarding the samples and the source images is quite similar. This information is passed to the reduce block for the final screening operation and is finally compared, to return the sample number and a similar number of points.

We observe the average time taken by the clusters of Raspberry Pi and also by the single PC, to evaluate performance.

5. Results and Discussion

The experiment is set up under IDE with an efficient map-reduce environment to output a java executable file which is then passed on to the Hadoop of the master host. Further, to set up the Hadoop environment, we pass the SSH command through the Linux terminal to confirm the connection between the nodes. Then we use the Linux terminal to issue other necessary instructions to operate Hadoop through the master node.

5.1 Hadoop Setup

The NameNode format instruction is run at the master node to compete for the Hadoop settings and count the number of nodes in the cluster through SSH communication so that the nodes can create a better HDFS folder. Moreover, to complete the process, the master node must start the decentralized file storage system.

After completing the above process, the image feature point samples are imported into Hadoop's decentralized storage system. First, we start the Hadoop HDFS by using start-dfs.sh in the master terminal. Then the nodes are connected through SSH, and the master can view the HDFS information via dfsadmin-report. Finally, the master node issues the hodoopdfs-put command to upload all the information to the distributed storage system.

The node terminal issued JPS instruction can see the operation of the NodeManager while the master terminal issued JPS instruction can see the operation of the ResourceManager. Further, this means that the Hadoop YARN system is active and running smoothly. We can also use the YARN browser interface to view the information of the cluster nodes and also observe the MapReduce jobs running through this interface.

5.2 Experimental Results

After completing the above Hadoop cluster setup, the MapReduce program is used to identify the feature points of the image and find the image similar to the input data, to test the overall performance. Table 6 shows the details about the characteristics of the HDFS data file.

Table 6. Details about the characteristics of the HDFS data file

Group	Data size	Information data test groups
1	251 MB	1,000
2	504.1MB	2,000
3	1 GB	4,000
4	2.01 GB	8,000
5	4.03 GB	16,000
6	8.07 GB	32,000
7	16.13 GB	64,000
8	32.26 GB	128,000

5.2.1 Single computer performance

We use i5-4440, 3.1 GHz CPU with 8 GB memory for this experimental setup and calculate the time consumed per 100 operations, as shown in Figs. 8, 9 and Table 7, using the total operation time. From the experimental results, we can observe that the amount of data does not affect processing efficiency, and the values of average time consumed are very close to each other.

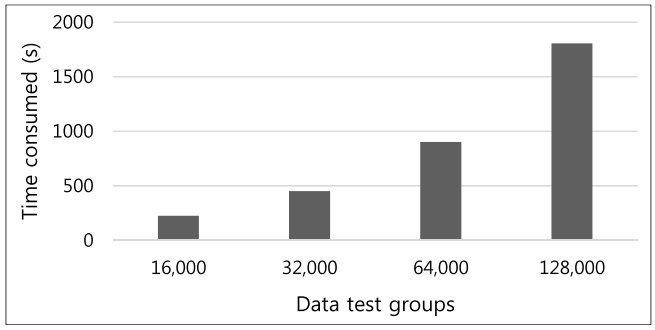


Fig. 8. Graph showing total average time consumed by a single computer for different classes of dataset.

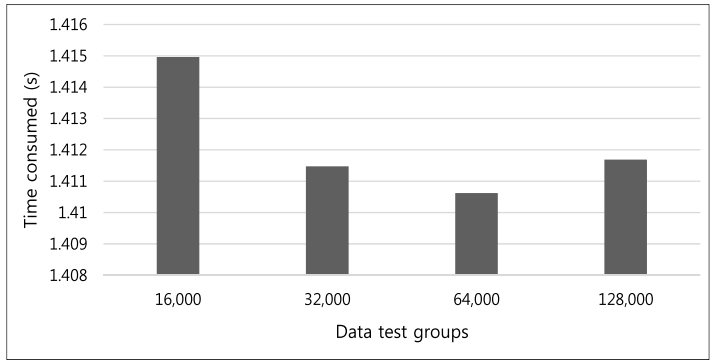


Fig. 9. Graph showing average time consumed by a single computer per 100 datasets for different classes of dataset.

Table 7. The average range of time consumed by a single computer per 100 data sets for each data test group

Data test groups	Time consumed per 100 datasets (s)
16,000	1.414±0.009
32,000	1.411±0.007
64,000	1.410±0.004
128,000	1.411±0.005

5.2.2 Raspberry Pi cluster performance

We set up 6 sets of Mobile Raspberry Pi Hadoop clusters, each containing 5, 6, 7, 8, 9, 10 sets of Raspberry Pi nodes. Similar to the single computer, we observe the changes in the volumes of data and their influence on the performance of the nodes for each cluster as shown in Figs. 10–15 and Tables 8–13.

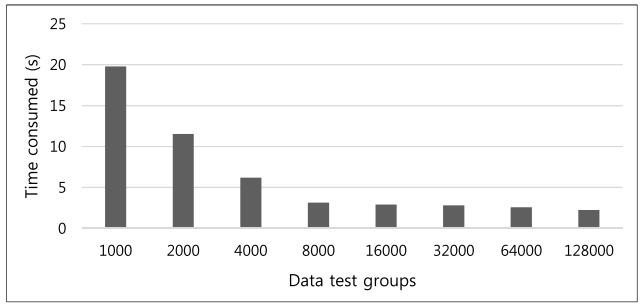


Fig. 10. Graph showing average time consumed by a 5 node cluster for different classes of dataset.

Table 8. The average range of time consumed by a 5 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	19.79 ± 3.68
2000	11.54 ± 1.55
4000	6.19 ± 0.12
8000	3.14 ± 0.13
16000	2.90 ± 0.12
32000	2.80 ± 0.11
64000	2.58 ± 0.08
128000	2.25 ± 0.11

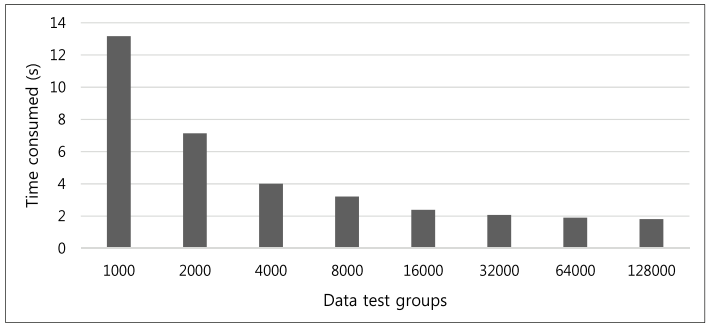


Fig. 11. Graph showing average time consumed by a 6 node cluster for different classes of data set.

Table 9. The average range of time consumed by a 6 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	13.16 ± 3.19
2000	7.13 ± 1.68
4000	4.01 ± 0.21
8000	3.21 ± 0.21
16000	2.39 ± 0.13
32000	2.06 ± 0.07
64000	1.89 ± 0.07
128000	1.80 ± 0.08

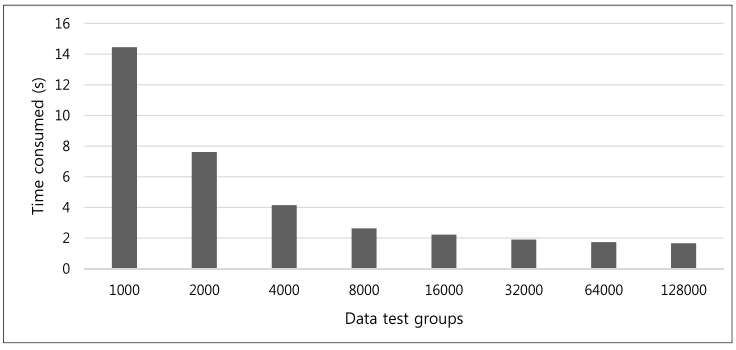


Fig. 12. Graph showing average time consumed by a 7 node cluster for different classes of dataset.

Table 10. The average range of time consumed by a 7 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	14.45 ± 2.36
2000	7.61 ± 1.22
4000	4.14 ± 0.29
8000	2.63 ± 0.28
16000	2.22 ± 0.04
32000	1.90 ± 0.05
64000	1.73 ± 0.02
128000	1.66 ± 0.03

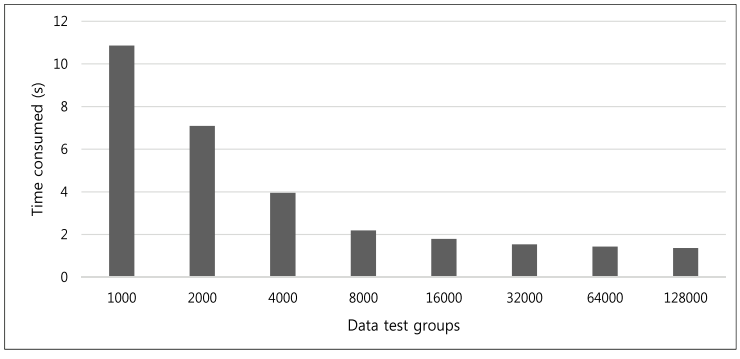


Fig. 13. Graph showing average time consumed by an 8 node cluster for different classes of dataset.

Table 11. The average range of time consumed by a 8 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	10.86 ± 2.74
2000	7.08 ± 1.23
4000	3.95 ± 0.37
8000	2.18 ± 0.13
16000	1.79 ± 0.10
32000	1.53 ± 0.07
64000	1.43 ± 0.06
128000	1.36 ± 0.03

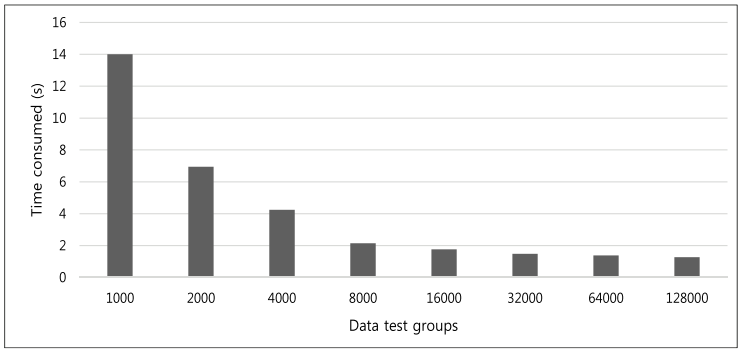


Fig. 14. Graph showing average time consumed by a 9 node cluster for different classes of dataset.

Table 12. The average range of time consumed by a 9 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	13.99 ± 3.20
2000	6.94 ± .156
4000	4.24 ± 0.26
8000	2.14 ± 0.09
16000	1.76 ± 0.19
32000	1.48 ± 0.09
64000	1.38 ± 0.12
128000	1.27 ± 0.03

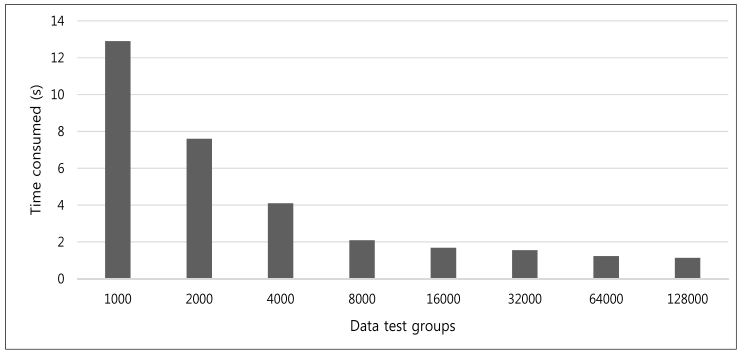


Fig. 15. Graph showing average time consumed by a 10 node cluster for different classes of dataset.

Table 13. The average range of time consumed by a 10 node cluster per 100 datasets for each data test group

Data test groups	Time consumed per 100 datasets (s)
1000	12.89 ± 3.30
2000	7.59 ± 1.08
4000	4.10 ± 0.20
8000	2.09 ± 0.14
16000	1.68 ± 0.13
32000	1.55 ± 0.35
64000	1.23 ± 0.04
128000	1.13 ± 0.05

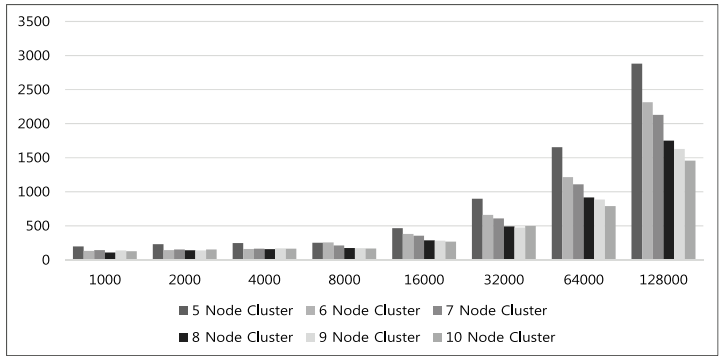


Fig. 16. Graph showing a comparison of total average time consumed by each node cluster for different classes.

We can observe in the Figs. 16 and 17 that the performance of a cluster with ten nodes is about 90% higher than that of the cluster with 5 nodes. The computation time decreases as the number of nodes increases. Further, this proves that Hadoop treats all nodes as individuals and does not share the individual resources. For example, with 128,000 data test groups, each additional cluster node will enhance the performance by 10% more than the previous arrangement. If the data is too small, increasing the number of nodes in the cluster will not significantly impact the computing efficiency.

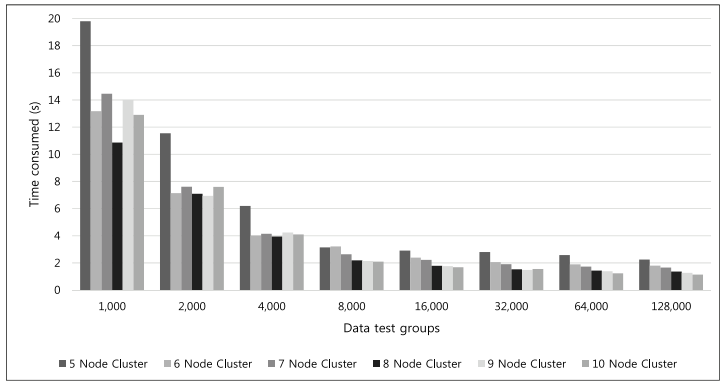


Fig. 17. Graph showing a comparison of average time consumed by each node cluster per 100 datasets for different classes of dataset.

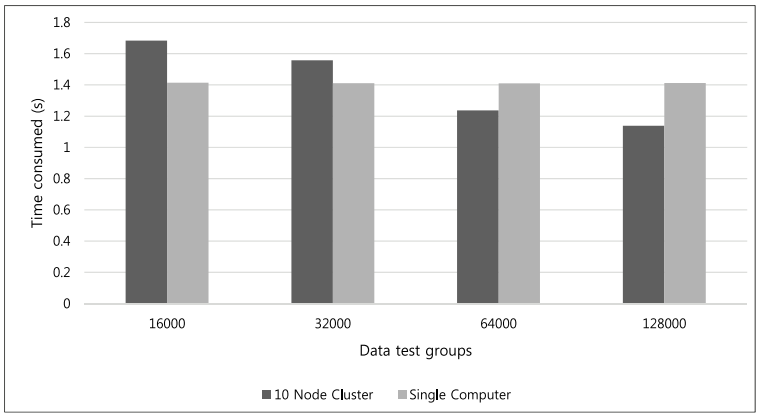


Fig. 18. Graph showing a comparison of total average time consumed by a 10 node cluster and by a single computer for different classes of dataset.

5.2.3 Performance comparison

We compare the computing performance for the single computer and the 10-node Raspberry Pi cluster for 128,000 data test groups. A single computer takes about 1.41 seconds to compute 100 feature information of an image whereas the Raspberry Pi cluster takes only 1.13 seconds. In a similar condition with 16,000 data test groups, a single computer takes about 1.4 seconds, while the raspberry pi cluster computes in 1.6 seconds. The detailed statistics can be seen in Fig. 18.

6. Conclusion

We use Apache Hadoop combined with the Raspberry Pi cluster to compute the big data generated via feature point extraction on an image. The MapReduce operation in Hadoop and the salient features of Raspberry Pi 3 helps us perform the experiment more efficiently compared to a high processing single computer, as shown in the study. However, the same cannot be said when we deal with a smaller dataset.

The Hadoop provides us the flexibility to quickly scale the setup for handling even more massive datasets. The HDFS storage system used by the Hadoop provides us with better security and higher fault tolerance when compared to single host storage. HDFS breaks the data into many blocks scattered over various nodes, which make it difficult to interpret the contents of information from a single node, thus providing better security for the data.

The low cost of the Raspberry Pi clusters and the scalable Hadoop environment can be used for a variety of complex operations in this era of big data.

References

[1] 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.

[2] J. M. Ross, "Roger Magoulas on big data," 2010 [Online]. Available: https://perma.cc/NXB5-ER87.

[3] P. Zikopoulos, C. Eaton, D. deRoos, T. Deutsch, and G. Lapis, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. New York, NY: McGraw-Hill, 2011.

[4] L. Xue, J. Ni, Y. Li, and J. Shen, "Provable data transfer from provable data possession and deletion in cloud storage," *Computer Standards & Interfaces*, vol. 54, pp. 46-54, 2017.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, 2004, pp. 137-150.

[6] C. Lam, "Introducing Hadoop," in *Hadoop in Action*. Stanford, CT: Manning Publications, 2011

[7] W. P. Birmingham and D. P. Siewiorek, "MICON: a knowledge based single board computer designer," in *Proceedings of the 21st Conference on Design Automation*, Albuquerque, NM, 1984, pp. 565-571.

[8] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, et al., "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment," in *Proceedings of 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, Bristol, UK, 2013, pp. 170-175.

[9] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'brien, "Iridis-Pi: a low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, no. 2, pp. 349-358, 2014.

[10] J. Kiepert, "Creating a raspberry pi-based Beowulf cluster," 2013 [Online]. Available: http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf.

[11] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The Glasgow raspberry pi cloud: a scale model for cloud computing infrastructures," in *Proceedings of 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Philadelphia, PA, 2013, pp. 108-112.

[12] N. Schot, "Feasibility of raspberry pi 2 based micro data centers in big data applications," in *Proceedings of the 23th University of Twente Student Conference on IT,* Enschede, The Netherlands, 2015, pp. 1-9.

[13] C. Kaewkasi and W. Srisuruk, "A study of big data processing constraints on a low-power Hadoop cluster," in *Proceedings of 2014 International Computer Science and Engineering Conference (ICSEC),* Khon Kaen, Thailand, 2014, pp. 267-272.

[14] B. Qureshi, Y. Javed, A. Koubaa, M. F. Sriti, and M. Alajlan, "Performance of a low cost Hadoop cluster for image analysis in cloud robotics environment," *Procedia Computer Science*, vol. 82, pp. 90-98, 2016.

[15] W. Hajji and F. P. Tso, "Understanding the performance of low power Raspberry Pi Cloud for big data," *Electronics*, vol. 5, no. 2, article no. 29, 2016.

[16] R. Morabito, "Virtualization on internet of things edge devices with container technologies: a performance evaluation," *IEEE Access*, vol. 5, pp. 8835-8850, 2017.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, 2008.

Kathiravan Srinivasan https://orcid.org/0000-0002-9352-0237

He received his B.E. in Electronics and Communication Engineering and M.E. in Communication Systems Engineering from Anna University, Chennai, India. He also secured a Ph.D. in Information and Communication Engineering from Anna University Chennai, India. He is currently Associate Professor in the School of Information Technology and Engineering at Vellore Institute of Technology, India. He earlier served as Deputy Director - Office of International Affairs and also as faculty in the Department of Computer Science and Information Engineering at

National Ilan University, Taiwan. He is presently an Associate Editor for IEEE Access and Editorial Board Member and reviewer for various SCI, SCIE and Scopus Indexed Journals. He has accomplished various globalization educational activities and partnerships. He has played an active role in organizing several international conferences, seminars, and lectures. He has been a keynote speaker at many international conferences and IEEE events.

Chuan-Yu Chang https://orcid.org/0000-0001-9476-8130

He received a Ph.D. in Electrical Engineering from National Cheng Kung University, Taiwan, in 2000. He is currently Distinguished Professor at the Department of Computer Science and Information Engineering and Dean of Research & Development, National Yunlin University of Science and Technology, Taiwan. His research interests include computational intelligence and its application to medical image processing, wafer defect inspection, emotion recognition, and pattern recognition. In the above areas, he has more than 150 publications in journals and conference proceedings. He serves as an Associate Editor for two international journals, *Multidimensional Systems and Signal Processing*, and *International Journal of Control Theory and Applications*. He is a Life Member of IPPR, TAAI, and a Senior Member of IEEE. He is an IET Fellow, the chair of IEEE Signal Processing Society, Tainan Chapter, and the representative for Region 10 of IEEE SPS Chapters Committee.

Chao-Hsi Huang https://orcid.org/0000-0002-5429-2534

He received his Ph.D. degree in Physics from National Taiwan University, Taiwan, R.O.C. in 2007. He is Associate Professor at the Institute of Computer Science and Information Engineering, National Ilan University. His research interests include RFID system, IOT, Cloud Computing, Mobile Device Programming, Parallel Computing and Science Computation.

Min-Hao Chang https://orcid.org/0000-0003-1730-4695

He received his M.E. in Computer Science and Information Engineering from National Ilan University, Yilan, Taiwan and B.E. in Computer Science and Information Engineering from Vanung University, Taoyuan, Taiwan. He currently serves as a Research Assistant at National Ilan University, Department of Chemical and Materials Engineering, Yilan, Taiwan. Also, he serves as an Administrative Assistant, National Ilan University, Department of Computer Science and Information Engineering, Taiwan.

Anant Sharma https://orcid.org/0000-0001-6392-8082

He received a B.Tech. in Computer Science Engineering from The LNM Institute of Information Technology, Jaipur, India in 2017. He has worked as a summer intern at ArcelorMittal, Kazakhstan and as a Research Assistant at National Ilan University, Taiwan. His current research interests include machine learning, computer vision, and evaluation of systems.

Avinash Ankur https://orcid.org/0000-0002-7605-9194

He competed his bachelor's in technology in the field of Computer Science from The LNM Institute of Information Technology, Jaipur, India. He worked as an intern at the Indian Institute of Information Technology, Patna, and as a Research Assistant at National Ilan University, Yilan, Taiwan. He has also been a part of the core organization committees of several national and international level college fests.