Cluster Comput (2017) 20:1101–1121
DOI 10.1007/s10586-017-0846-z

The implementation of a cloud city traffic state assessment system using a novel big data architecture

Chao-Tung Yang[1] · Shuo-Tsung Chen[1] · Yin-Zhen Yan[1]

Received: 15 August 2016 / Revised: 28 January 2017 / Accepted: 28 March 2017 / Published online: 5 April 2017

Abstract In order to store and analyze the increasing data in recent years, big data techniques are applied to many fields such as healthcare, manufacturing, telecommunications, retail, energy, transportation, automotive, security, environment, etc. This work implements a city traffic state assessment system in cloud using a novel big data architecture. The proposed system provides the real-time busses location and real-time traffic state, especially the real-time traffic state nearby, through open data, cloud computing, bid data technology, clustering methods, and irregular moving average. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture is first implemented successfully and efficiently. Next, we utilize irregular moving average and clustering methods to find the area of traffic jam. Finally, three important experiments are performed. The first experiment indicates that the computing ability of Spark is better than that of Hadoop. The second experiment applies Spark to process bus location data under different number of executors. In the last experiment, we apply irregular moving average and clustering methods to efficiently find the area of traffic jam in Taiwan Boulevard which is the main road in Taichung city. Based on these experimental results, the provided system services are present via an advanced web technology.

✉ Chao-Tung Yang
ctyang@thu.edu.tw

Shuo-Tsung Chen
shough33@thu.edu.tw

Yin-Zhen Yan
r541754175@gmail.com

1 Department of Computer Science, Tunghai University, Taichung City 40704, Taiwan, ROC

Keywords Cloud · Traffic state assessment · Big data · Fuzzy C-means · Irregular moving average · Spark · HBase

1 Introduction

In recent years, the amount of data is getting bigger so that the processing and analysis of data have become more complex. It is more hard to obtain useful information from the data. Big data technique is an useful and powerful solution [1–3]. Now its applications are applied to many fields such as healthcare, manufacturing, telecommunications, retail, energy, transportation, automotive, security, environment, etc. To conform the growth of Big Data applications, almost all major industries and companies in the world invested a lot of time and money in developing Big Data tecnology. Based on the Big Data technology, these industries and companies can analyze and process the massive data they have and finally come out the valuable information. IBM predicted that global data amount continue to grow rapidly. It is predicted that the data amount would breakthrough 8000 Exabyte (EB, 1EB = 1 Million Terabyte) in 2015. Data analysis is a quite common concept in life after collecting all the data, and then the analysis will come out the results to be the basis of future action and decision [4].

Accordingly, lage amount of big data techniques and cloud techniques are proposed. Barbierato et al. [5] adopted NoSQL to store big data since NoSQL provides a mechanism for storage and retrieval of data that is better than the tabular relations used in relational databases. Zhang et al. [6] and Yang et al. [7,8] used HBase to store big data since HBase provides the distributed data storage cluster through HDFS in Hadoop. Their experiments indicate a good performance. Thus, both NoSQL and HBase have advantages in storing big data. Gu et al. [9] applied Hadoop MapReduse to process big data successfully. However, for processing real-time big data, Hadoop MapReduse is unable to show superiority due to the fact that Hadoop MapReduse needs more time on starting JOB and then distribute JOB to each node. To improve this drawback, the authors adopted the IN memory of Spark to achieve high-speed computation since Sparks in-memory primitives provide performance up to 100 times faster for certain applications. In addition, Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including Hadoop Distributed File System (HDFS), Cassandra, OpenStack Swift, and Amazon S3. Thus, our proposed architecture will combine Spark with the distribution computation of Hadoop YARN to enhance performance [10].

In order to analyze and improve the public transport, a government utilizes GPS positioning to record the relevant map location of most urban public transport systems, coupled with the back-end processing of data transmission via GPRS or 3G to track the transport status [11,12]. Figure 1 shows a bus positioning and data transmission scheme. Many cities provide these system services for users to know the bus location and estimate waiting time [13]. However, city traffic state is worsening due to the economic development and population growth. Zeng et al. proposed a novel multi-sensor traffic state assessment system based on incomplete data [14]. Their system comprises probe vehicle detection sensors, fixed detection sensors, and traffic state assessment algorithm. The results show that their system is effective to assess traffic state, and it is suitable for the urban intelligent transportation system. However, the analysis efficiency and storage of real-time data is not sufficient. It is necessary to combine big data architecture with cloud computing to solve the challenges including big data capture, big data storage, big data analysis, parallel computing, etc.

This work presents a cloud city traffic state assessment system using a novel big data architecture. The proposed system provides the real-time busses location and real-time

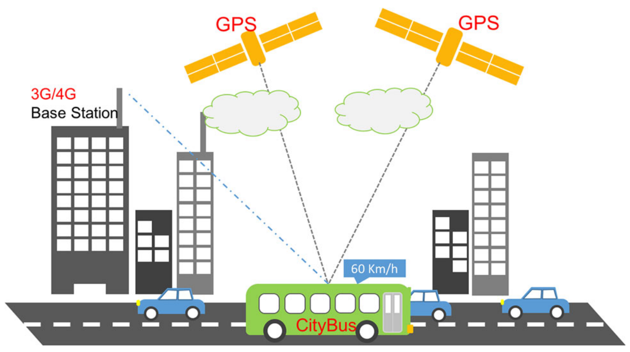


Fig. 1 Bus positioning and data transmission schematic

traffic situation, especially the real-time traffic situation nearby, through open data, GPS, GPRS and cloud technologies. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture is first implemented successfully and efficiently. Next, we utilize moving average and clustering methods to find the area of traffic jam. Finally, two important experiments are performed. The first experiment indicates that the computing ability of Spark is better than that of Hadoop. In the second experiment, we apply moving average and clustering methods including DBSCAN, K-means, and Fuzzy C-means to efficiently find the area of traffic jam in Taiwan Boulevard which is the main road in Taichung city. Based on these experimental results, the provided system services are present via an advanced web technology.

The rest of the paper is as follows. Section 2 reviews some mathematical preliminaries. In Sect. 3, we introduce the proposed system design and implementation. Section 4 shows our experiment environment and results. In Sect. 5, some conclusions are given.

2 System architecture design and implementation

In this section, the proposed cloud city traffic state assessment system collects open data to provide real-time traffic state, especially the real-time traffic state nearby. Because the real-time data collected is huge and from different attributes, the proposed system first utilizes a novel cloud architecture of big data to store, process, and analyze a huge amount of real-time data and then provides useful traffic information and services by using irregular moving average and clustering. In addition, the proposed system also store the generated real-time information to be historical data for the improvement of the system accuracy.

2.1 Proposed cloud architecture of big data

The proposed system architecture, as shown in Fig. 4, is introduced in this section. First of all, we collect open data including real-time data and historical data from government server. The capture details are as follows: In real-time open data including bus dynamic location and weather information,we utilize Python Program to capture these real-time data from government server and then store these data in a file system HDFS managed by Big Data Distributed Database, Hbase, through a tool Hbase API, as shown in Fig. 2. Generally, historical data including bus GPS data is huge and with different format. To solve these two problems, we first utilize Python Program to capture these data respectively and then process the different format of these data by Apache Spark. Finally, the processed data are stored in Hbase. Figure 3 shows the history data collection.

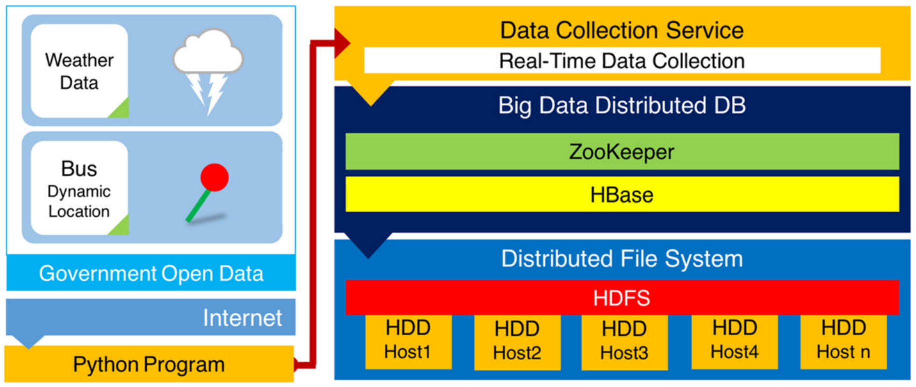


Fig. 2 Real time data collection service

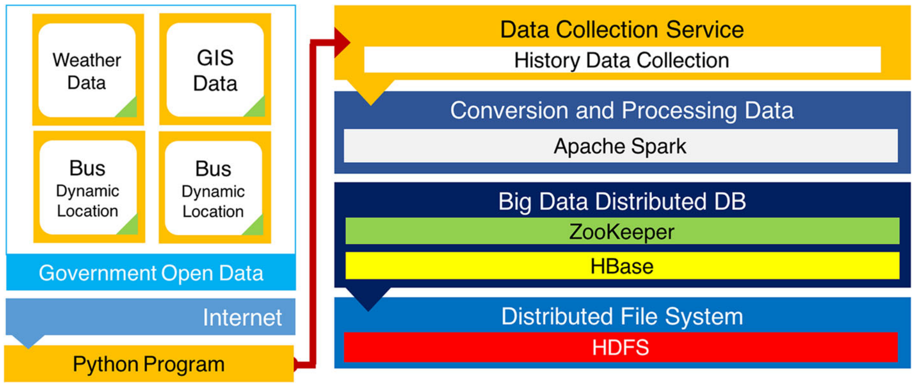


Fig. 3 History data collection service

Next, the data in HBase is transferred to Cloud Storage and processed based on the high-scalability cloud technologies [15–17], Hadoop [18] and Spark [19]. In data storage, we used Hadoop HDFS to be a cloud storage basis and then a NoSQL database for big data, namely HBase, is utilized to establish the cluster of distributed data storage including structured and unstructured data based on the Hadoop HDFS. In data analysis and computation, we adopted Spark to meet the requirement of the high-speed real-time computation since Spark can quickly assess and analyze the data stored in HBase. Finally, these results of analyzing real-time traffic situation is friendly present by web application service through several web technologies, java scritp, html5, and css3 combined with d3.jspquery. Figure 5 is the flowchart of the proposed cloud city traffic state assessment system.

2.2 System implementation

Fourteen nodes were used to build a cloud cluster platform by using Cloudera Manager, Two nodes as master, Twelve node as the computing node to set up Apache ZooKeeper 3.4.5, Apache Hadoop HDFS 2.5.0, Apache Hadoop YARN 2.5.0, Apache HBase 0.98.6, and Apache Spark 1.2.0.

2.3 Cluster deployment

On the deployment, platform environment using two servers as master, and using 10 Gigabit Ethernet connection. computing nodes using 1 Gigabit Ethernet, each node as DataNode, NodeManage, and RegionServer, where three computing nodes as ZooKeeper, as shown in Fig. 6.

Cloudera Manager is used to monitor service states and system loading, in service states such as Spark, HBase, HDFS, YARN and ZooKeeper service status, or in system loading such as CPU usage, Memory usage, Disk I/O, network and Disk usage.

Cloudera Manager can also monitor the status of each node, confirming normal connections of each host. Cloudera Manager checks at regular intervals, and it will warn if connections are abnormal or the connection quality is poor. Cloudera Manager can remove nodes at any time, adding or removing nodes into or out of the cluster.

Through the Fourteen hosts, i.e., the two NameNode and Twelve DataNodes, the Hadoop HDFS NameNode Web Interface shows that the cluster provides 10.47 TB of big data storage space. This information also shows how many live DataNodes are functioning shown in Fig. 7.

Fig. 4 System architecture

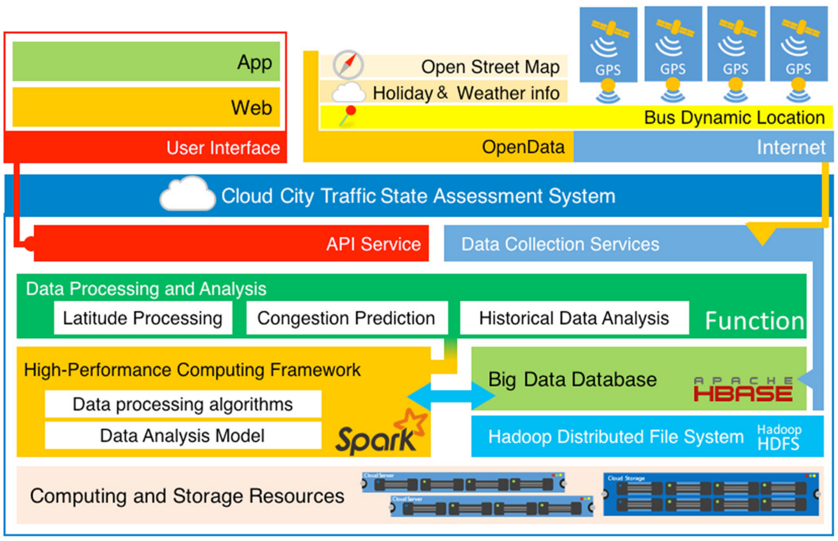
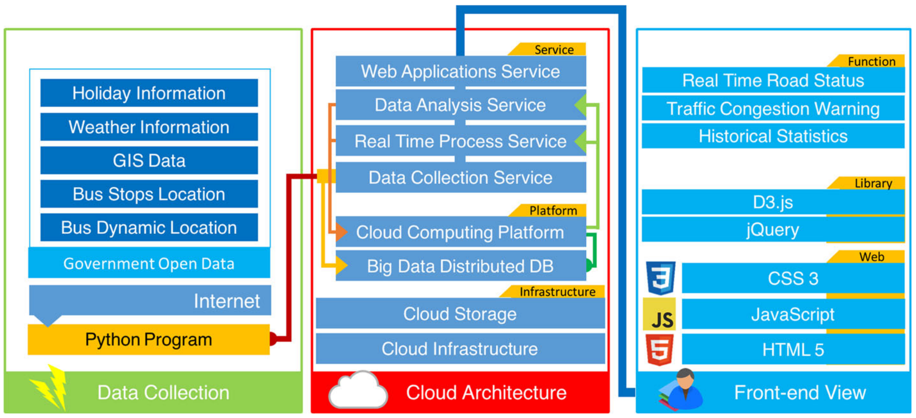


Fig. 5 Flowchart of the proposed cloud city traffic state assessment system



2.4 Implementation of the provided system services

In Taichung City Bus Dynamic System, there are 245 bus routes and over 1000 bus drive in these route at rush hour. Taichung City Bus Dynamic System provides each bus information in XML format including bus position, bus number, bus route, bus velocity, and so on. In addition, XML format is shown in Fig. 9 the Taichung City Bus Dynamic System updates each bus position every 20 s. For an instance, this thesis implement the proposed Cloud City Traffic State Assessment System in Taiwan Boulevard Taichung, Taiwan. First of all, the proposed system used python programming to captures the bus position updating data per second into our storage basis in Hadoop HBase. In this step, the host road is randomly divide into several blocks according to the intersection with other roads is shown in Figs. 8 and 10. Then, we apply Fuzzy C-means clustering mentioned in section Backgrounds and preliminaries to roughly classify the blocks and their size. Finally, the real-time traffic state in three levels, Jam, Normal, and Smooth for each block is evaluated by irregular moving average which is introduced as follow.

Suppose N_t is the number of bus at time t and

$$S = \left\{x_i, x_{i+1}, x_{i+2}, \ldots, x_{N_t}\right\} \tag{1}$$

is a subset of sample values, each of which denotes the average time of a bus within ith block. Then, a new series of

$$\{A_1, A_2, \ldots, A_i, \ldots\} \tag{2}$$

is called the irregular moving average of S which is obtained by the following calculation:

$$A_i = \frac{x_i + x_{i+1} + x_{i+2} + \cdots + x_{N_t}}{N_t} \tag{3}$$

3 Performance comparison for different techniques

In this section, we have two performance comparisons for later use. The first is the computation ability of Spark and Hadoop. The second is the efficiency of three clustering methods,

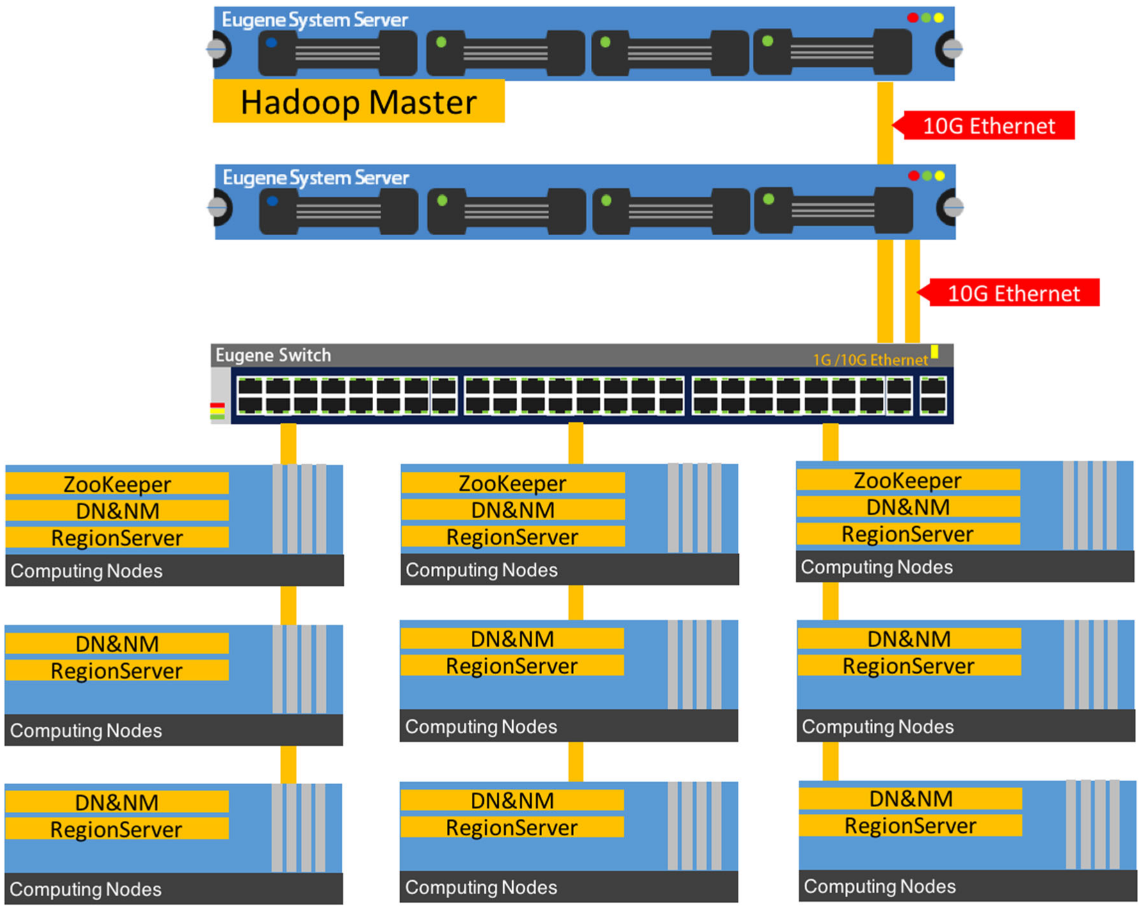


Fig. 6 System deployment architecture diagram

3.1 Spark and Hadoop MapReduce performance comparison

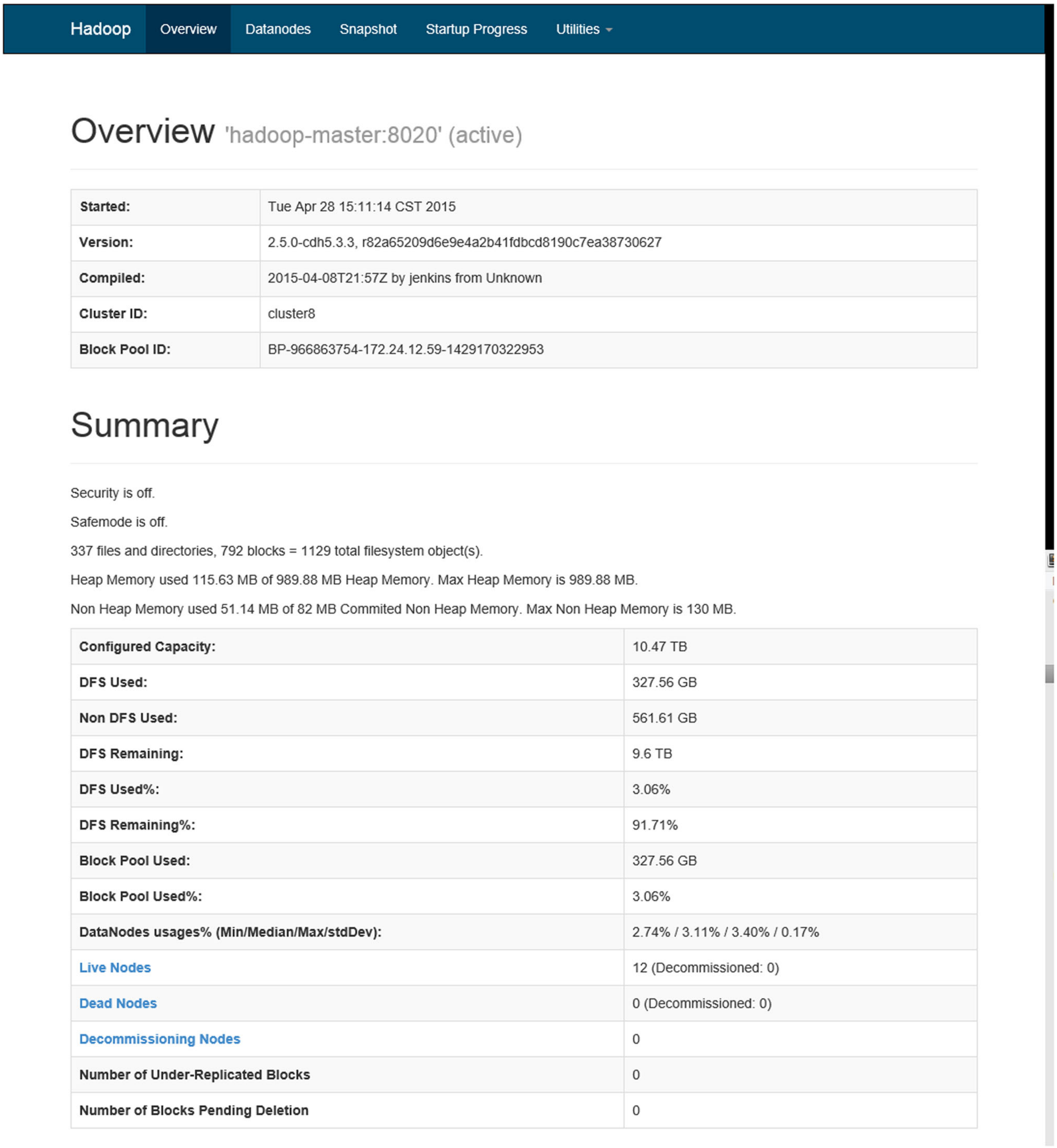
To verify that Spark adopted in our system has better performance than Hadoop MapReduce, this subsection gives a comparison test for Spark and Hadoop using WordCount programming. First, ten test files of size 1GB 10GB are randomly generated and then put these files into HDFS. Next, the ten files in HDFS are executed by Spark and Hadoop individually. As shown in Fig. 11, Spark cost less time regardless of file size. It is worth mentioning that the difference between them increases when the test file size increases.

Data processing and analysis are important to the proposed system. Before data analysis, the proposed system needs to read and write data. In other words, data read and data write are two important parts in data processing. Since the proposed system store data in HDFS, we adopt Hadoop TESTDFSIO to test the read and write performance by changing replication number and MAP number in different data size to obtain the best solution. The detail is as follows. First of all, we test data read speed and data write speed in three data sizes, 12G, 60G, and 120G. In each data size, we increase the replication number from 1 to 12 to observe the data read speed and data write speed. At the same time, we adjust TESTDFSIO arguments to observe the data read speed and data write speed under various MAP number which is a multiple of number of cluster nodes, especially (1–5) number of nodes. Since TESTDFSIO is a MapReduce program, the test procedure is given as follows and as shown in Fig. 12:

- Step 1. Input user parameters in beginning.
- Step 2. Arrange the number of MAP and the data size for each MAP when reading and writing data.
- Step 3. Read and write data into node by MAP.
- Step 4. Estimate MAP execution time and collect corresponding results.
- Step 5. Output the results.

In the results. Figure 13 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 12GB.

Figure 14 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 60GB.



Overview 'hadoop-master:8020' (active)

Started:	Tue Apr 28 15:11:14 CST 2015
Version:	2.5.0-cdh5.3.3, r82a65209d6e9e4a2b41fdbcd8190c7ea38730627
Compiled:	2015-04-08T21:57Z by jenkins from Unknown
Cluster ID:	cluster8
Block Pool ID:	BP-966863754-172.24.12.59-1429170322953

Summary

Security is off.

Safemode is off.

337 files and directories, 792 blocks = 1129 total filesystem object(s).

Heap Memory used 115.63 MB of 989.88 MB Heap Memory. Max Heap Memory is 989.88 MB.

Non Heap Memory used 51.14 MB of 82 MB Commited Non Heap Memory. Max Non Heap Memory is 130 MB.

Configured Capacity:	10.47 TB
DFS Used:	327.56 GB
Non DFS Used:	561.61 GB
DFS Remaining:	9.6 TB
DFS Used%:	3.06%
DFS Remaining%:	91.71%
Block Pool Used:	327.56 GB
Block Pool Used%:	3.06%
DataNodes usages% (Min/Median/Max/stdDev):	2.74% / 3.11% / 3.40% / 0.17%
Live Nodes	12 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0

Fig. 7 Hadoop NameNode information

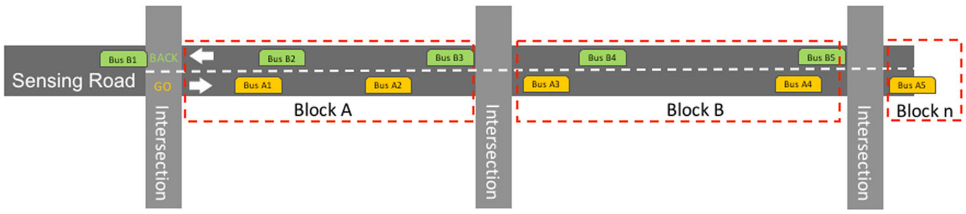
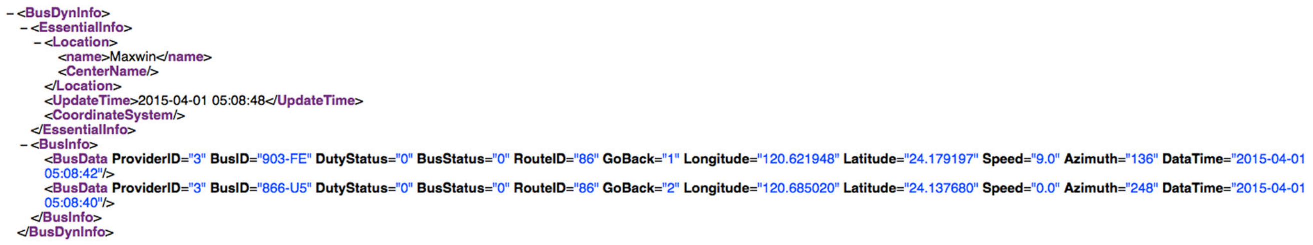


Fig. 8 Block division schematic



```
– <BusDynInfo>
  – <EssentialInfo>
    – <Location>
        <name>Maxwin</name>
        <CenterName/>
      </Location>
      <UpdateTime>2015-04-01 05:08:48</UpdateTime>
      <CoordinateSystem/>
    </EssentialInfo>
  – <BusInfo>
      <BusData ProviderID="3" BusID="903-FE" DutyStatus="0" BusStatus="0" RouteID="86" GoBack="1" Longitude="120.621948" Latitude="24.179197" Speed="9.0" Azimuth="136" DataTime="2015-04-01
      05:08:42"/>
      <BusData ProviderID="3" BusID="866-U5" DutyStatus="0" BusStatus="0" RouteID="86" GoBack="2" Longitude="120.685020" Latitude="24.137680" Speed="0.0" Azimuth="248" DataTime="2015-04-01
      05:08:40"/>
    </BusInfo>
  </BusDynInfo>
```

Fig. 9 XML format

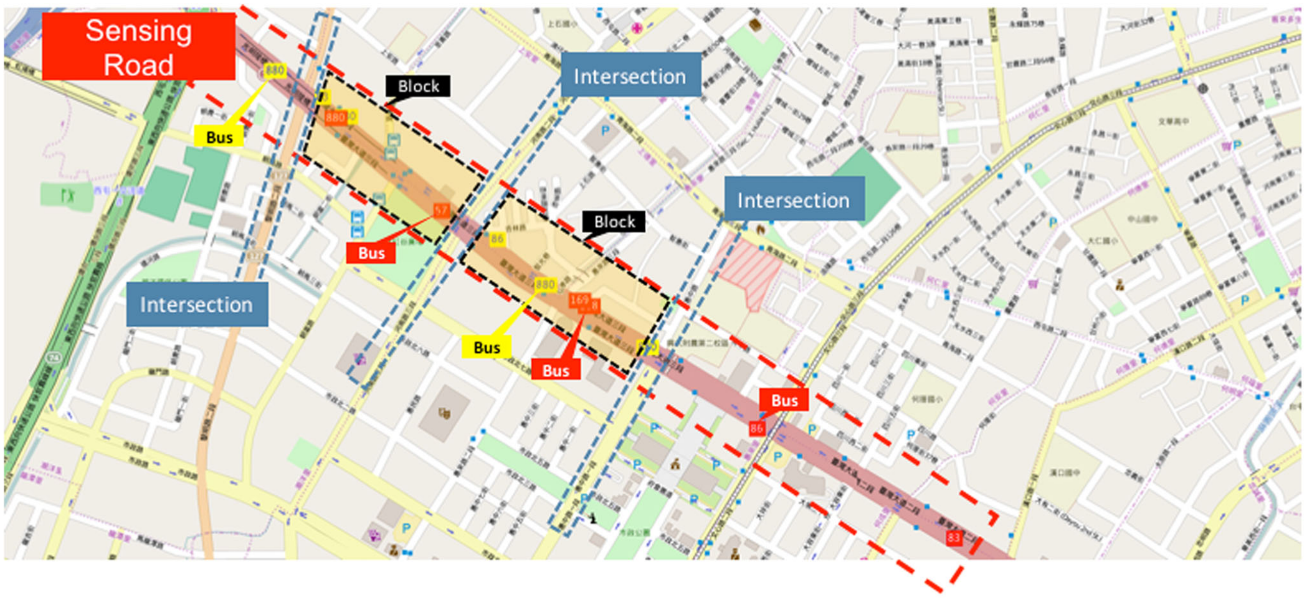
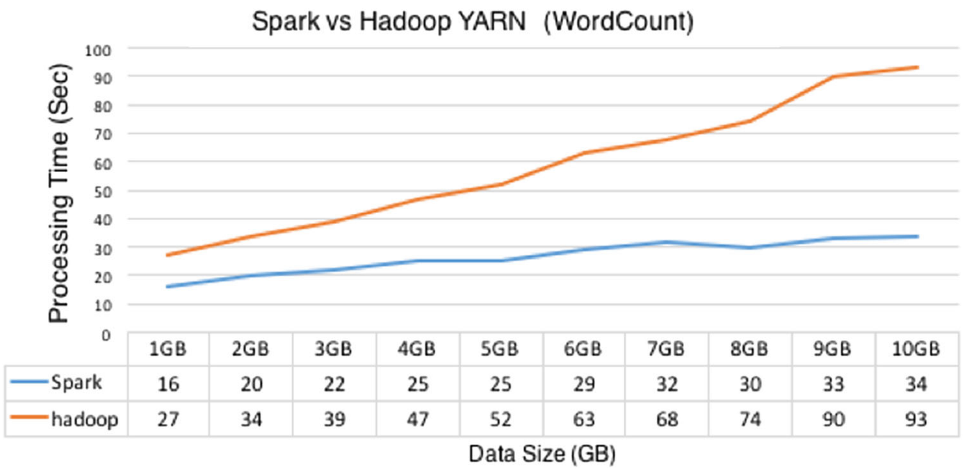


Fig. 10 Block division schematic apply to Taiwan Boulevard

Fig. 11 Spark and Hadoop MapReduce performance comparison



	1GB	2GB	3GB	4GB	5GB	6GB	7GB	8GB	9GB	10GB
Spark	16	20	22	25	25	29	32	30	33	34
hadoop	27	34	39	47	52	63	68	74	90	93

Figure 15 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 120GB. From above figures, one can observe that more replication number more writing time under various data size and various MAP but reading time slightly decreases when replication number increases, as shown in Fig. 16. In conclusion, replication 2 has the best result. In the point of view on different test size, writing time increases stably but reading time decreases unstably, as shown in Fig. 17. In addition, one can observe that both reading time and writing time are short for bigger data file. It means once reading or writing a big data file is efficient.

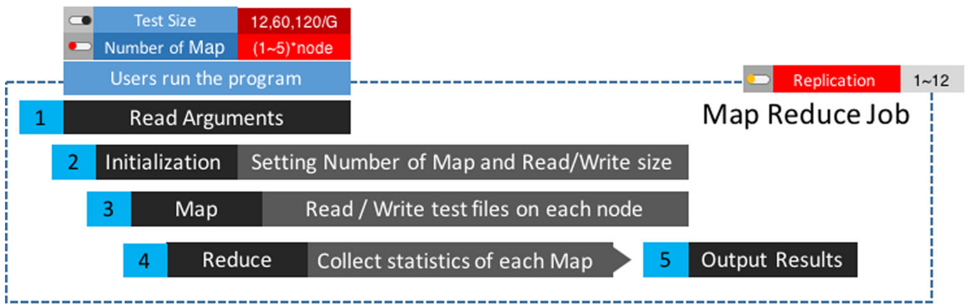
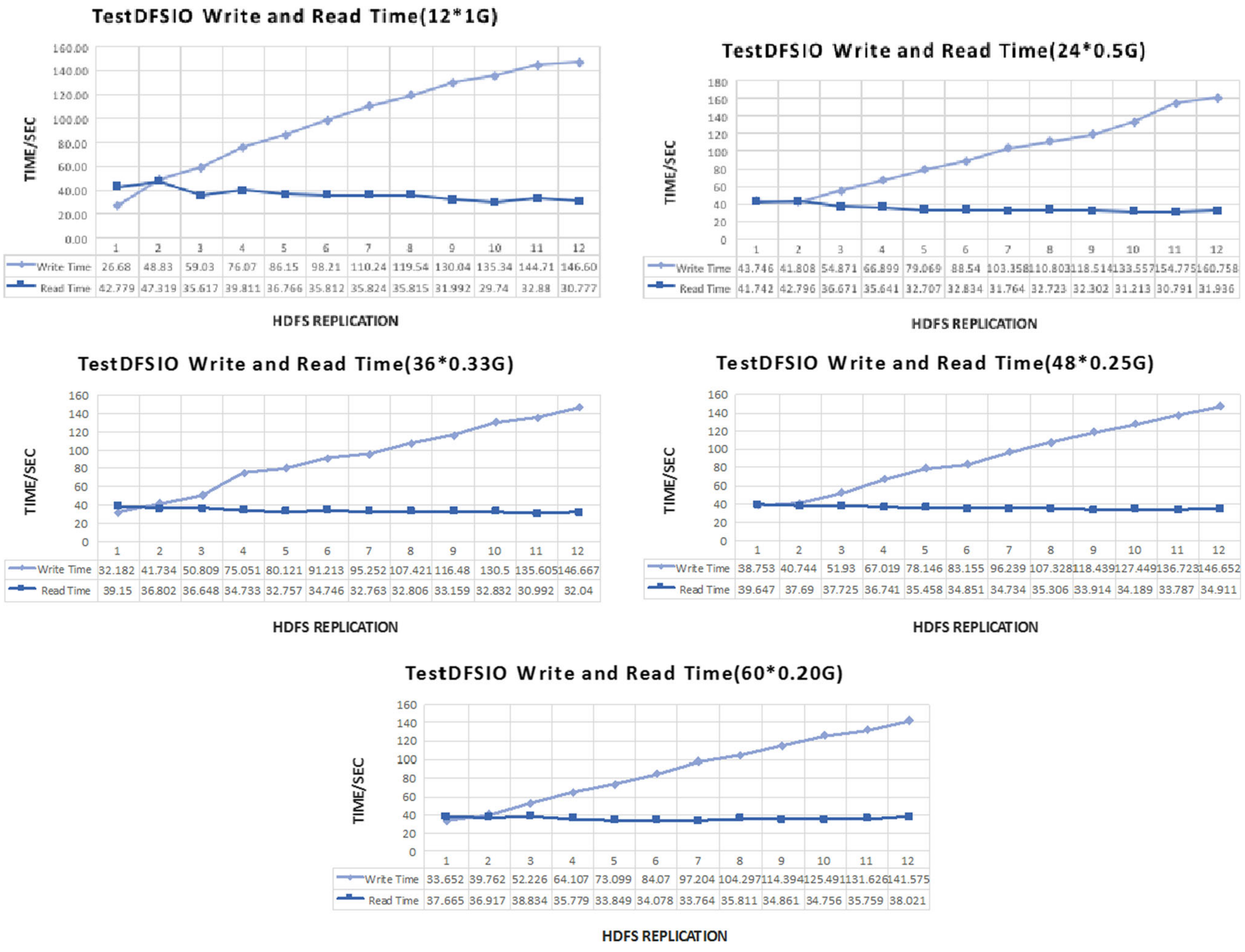


Fig. 12 The flow chart of HDFS read and write test



TestDFSIO Write and Read Time(12*1G)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	26.68	48.83	59.03	76.07	86.15	98.21	110.24	119.54	130.04	135.34	144.71	146.60
Read Time	42.779	47.319	35.617	39.811	36.766	35.812	35.824	35.815	31.992	29.74	32.88	30.777

TestDFSIO Write and Read Time(24*0.5G)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	43.746	41.808	54.871	66.899	79.069	88.54	103.358	110.803	118.514	133.557	154.775	160.758
Read Time	41.742	42.796	36.671	35.641	32.707	32.834	31.764	32.723	32.302	31.213	30.791	31.936

TestDFSIO Write and Read Time(36*0.33G)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	32.182	41.734	50.809	75.051	80.121	91.213	95.252	107.421	116.48	130.5	135.605	146.667
Read Time	39.15	36.802	36.648	34.733	32.757	34.746	32.763	32.806	33.159	32.832	30.992	32.04

TestDFSIO Write and Read Time(48*0.25G)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	38.753	40.744	51.93	67.019	78.146	83.155	96.239	107.328	118.439	127.449	136.723	146.652
Read Time	39.647	37.69	37.725	36.741	35.458	34.851	34.734	35.306	33.914	34.189	33.787	34.911

TestDFSIO Write and Read Time(60*0.20G)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	33.652	39.762	52.226	64.107	73.099	84.07	97.204	104.297	114.394	125.491	131.626	141.575
Read Time	37.665	36.917	38.834	35.779	33.849	34.078	33.764	35.811	34.861	34.756	35.759	38.021

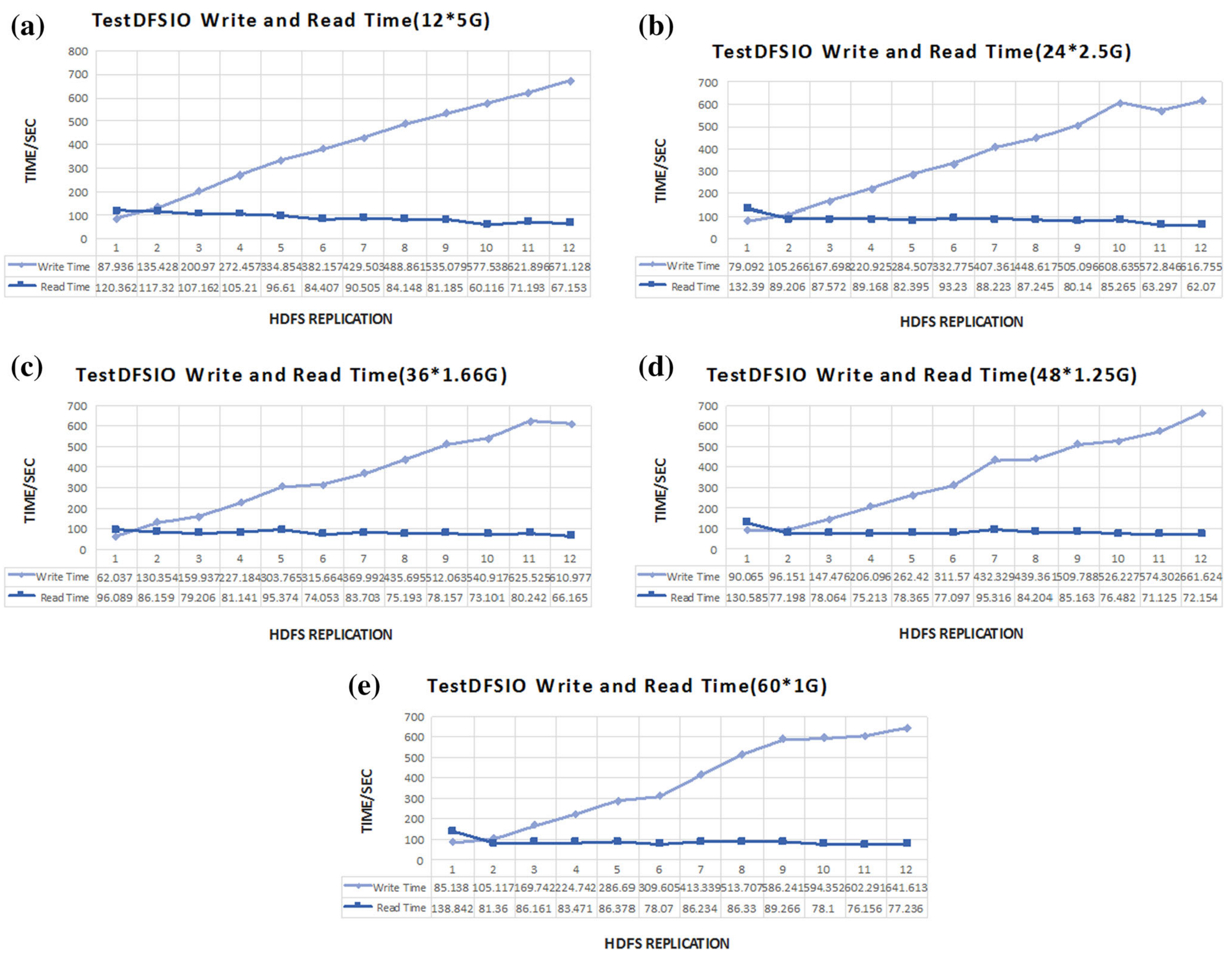
Fig. 13 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 12GB

3.2 Using Spark to process bus location data under different number of executors

Most input data of the proposed system is obtained through the Government OPEN DATA. These input data are collected every two seconds from open data. The collected data are about 5GB one day. They will be a big historical data. Accordingly, the processing and calculation of these big data are achieved through Spark Application. To process and analyze the big data efficiently, two Spark Applications are test. Each Spark Application has 12 executors for testing one-day, two-day, four-day data and six kinds of memory size.

The first Spark Application, namely convBus, is mainly to remove unwanted data in BUS GPS information and duplicate data in accordance with the update time so as to find the Block of the given latitude and longitude coordinates. As shown in Fig. 18, the detail procedure is summarized as follows:

- Step 1. Read BUS record data file.



(a)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	87.936	135.428	200.97	272.457	334.854	382.157	429.503	488.861	535.079	577.538	621.896	671.128
Read Time	120.362	117.32	107.162	105.21	96.61	84.407	90.505	84.148	81.185	60.116	71.193	67.153

(b)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	79.092	105.266	167.698	220.925	284.507	332.775	407.361	448.617	505.096	608.635	572.846	616.755
Read Time	132.39	89.206	87.572	89.168	82.395	93.23	88.223	87.245	80.14	85.265	63.297	62.07

(c)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	62.037	130.354	159.937	227.184	303.765	315.664	369.992	435.695	512.063	540.917	625.525	610.977
Read Time	96.089	86.159	79.206	81.141	95.374	74.053	83.703	75.193	78.157	73.101	80.242	66.165

(d)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	90.065	96.151	147.476	206.096	262.42	311.57	432.329	439.361	509.788	526.227	574.302	661.624
Read Time	130.585	77.198	78.064	75.213	78.365	77.097	95.316	84.204	85.163	76.482	71.125	72.154

(e)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	85.138	105.117	169.742	224.742	286.69	309.605	413.339	513.707	586.241	594.352	602.291	641.613
Read Time	138.842	81.36	86.161	83.471	86.378	78.07	86.234	86.33	89.266	78.1	76.156	77.236

Fig. 14 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 60GB. **a** Map = 12, **b** map = 24, **c** map = 36, **d** map = 48, **e** map = 60

– Step 2. Filter necessary BUS information and remove XML format data.
– Step 3. Use MAP to group Update Time and cluster coordinate into proper Block.
– Step 4. Remove duplicate data.
– Step 5. Output data to HDFS.

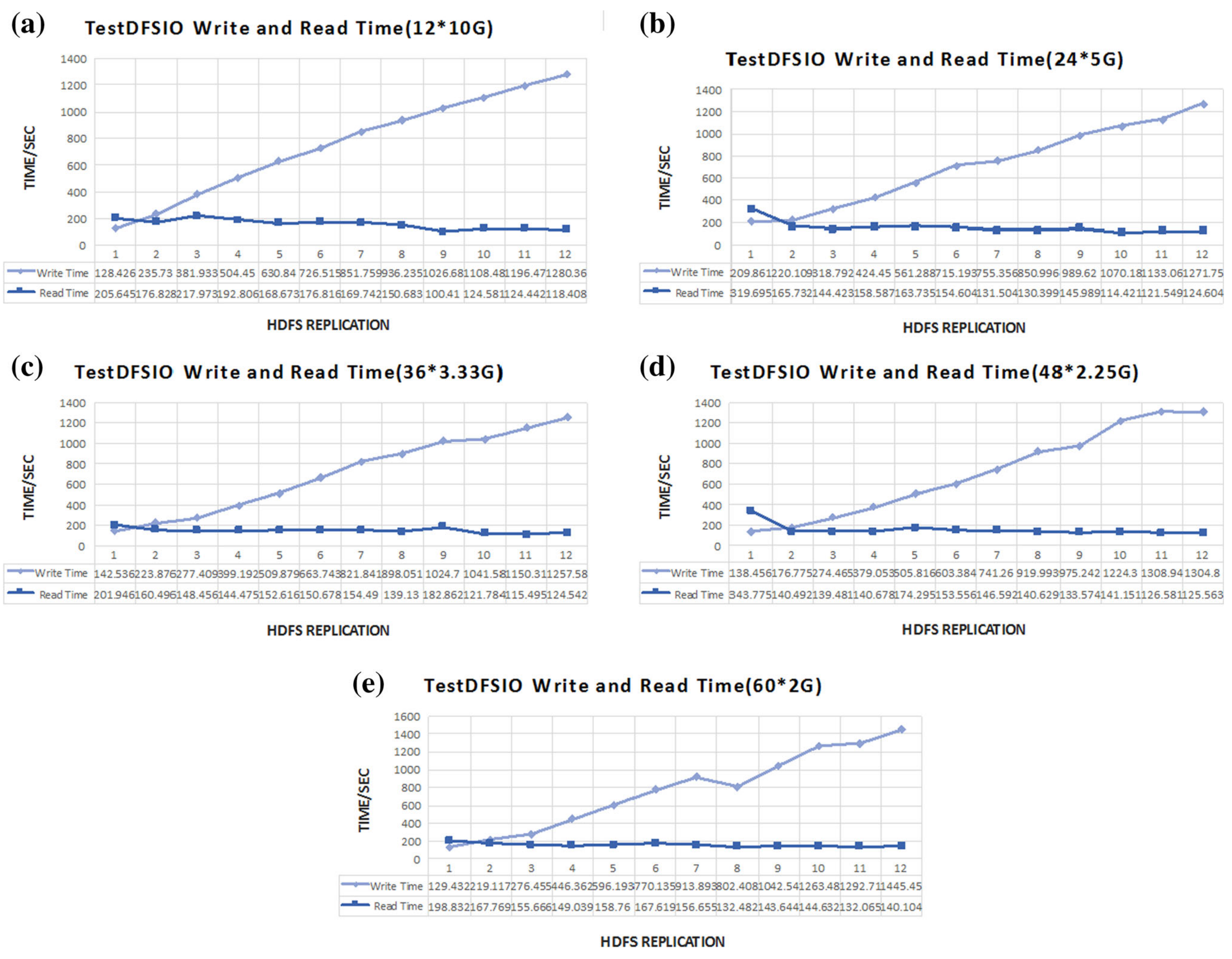
The second Spark Application, namely comBus, has almost the same procedures with the first Spark Application. The difference between them is that comBus has a permutation for Block in last two steps. As shown in Fig. 19, the detail procedure is summarized as follows:

– Step 1. Read BUS record data file.
– Step 2. Filter necessary BUS information and remove XML format data.
– Step 3. Use MAP to group Update Time and cluster coordinate into proper Block.
– Step 4. Remove duplicate data.
– Step 5. Sort by Block.
– Step 6. Output data to HDFS.

Figures 20, 21 and 22 show the execution time of using both convBus and comBus to process one-day, two-day and three-day data under different number of executors and different memory. One can observe that processing time decreases when executors increase under fixed memory. However, processing time is similar when memory increases, especially when executors are greater than three. We also observe that the execution time of comBus is greater than the execution time of convBus. To reduce their processing time, increase of the number of executors is a consideration.

3.3 Comparison of different clustering methods to find traffic jams

This section first finds traffic jams by three popular clustering methods: DBSCAN, K-means, Fuzzy-C-Means. Next,



(a)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	128.426	235.73	381.933	504.45	630.84	726.515	851.759	936.235	1026.68	1108.48	1196.47	1280.36
Read Time	205.645	176.828	217.973	192.806	168.673	176.816	169.742	150.683	100.41	124.581	124.442	118.408

(b)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	209.861	220.109	318.792	424.45	561.288	715.193	755.356	850.996	989.62	1070.18	1133.06	1271.75
Read Time	319.695	165.732	144.423	158.587	163.735	154.604	131.504	130.399	145.989	114.421	121.549	124.604

(c)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	142.536	223.876	277.409	399.192	509.879	663.743	821.841	898.051	1024.7	1041.58	1150.31	1257.58
Read Time	201.946	160.496	148.456	144.475	152.616	150.678	154.49	139.13	182.862	121.784	115.495	124.542

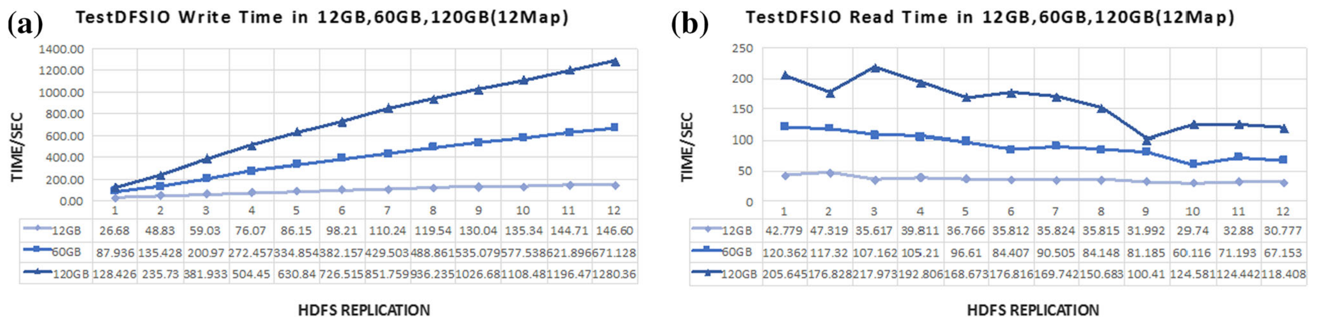
(d)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	138.456	176.775	274.465	379.053	505.816	603.384	741.26	919.993	975.242	1224.3	1308.94	1304.8
Read Time	343.775	140.492	139.481	140.678	174.295	153.556	146.592	140.629	133.574	141.151	126.581	125.563

(e)

	1	2	3	4	5	6	7	8	9	10	11	12
Write Time	129.432	219.117	276.455	446.362	596.193	770.135	913.893	802.408	1042.54	1263.48	1292.71	1445.45
Read Time	198.832	167.769	155.666	149.039	158.76	167.619	156.655	132.482	143.644	144.632	132.065	140.104

Fig. 15 Read and write time duration for MAP = 12, 24, 36, 48, 60 in various replication number under the case 120GB. **a** Map = 12, **b** map = 24, **c** map = 36, **d** map = 48, **e** map = 60



(a)

	1	2	3	4	5	6	7	8	9	10	11	12
12GB	26.68	48.83	59.03	76.07	86.15	98.21	110.24	119.54	130.04	135.34	144.71	146.60
60GB	87.936	135.428	200.97	272.457	334.854	382.157	429.503	488.861	535.079	577.538	621.896	671.128
120GB	128.426	235.73	381.933	504.45	630.84	726.515	851.759	936.235	1026.68	1108.48	1196.47	1280.36

(b)

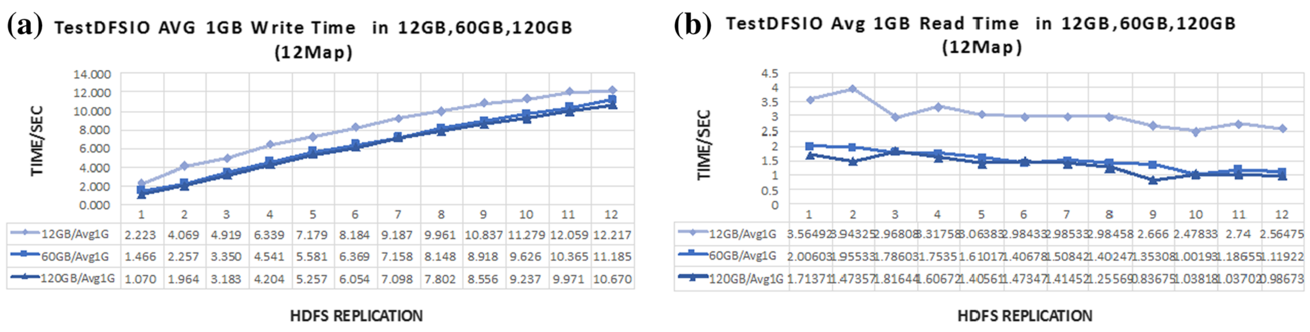
	1	2	3	4	5	6	7	8	9	10	11	12
12GB	42.779	47.319	35.617	39.811	36.766	35.812	35.824	35.815	31.992	29.74	32.88	30.777
60GB	120.362	117.32	107.162	105.21	96.61	84.407	90.505	84.148	81.185	60.116	71.193	67.153
120GB	205.645	176.828	217.973	192.806	168.673	176.816	169.742	150.683	100.41	124.581	124.442	118.408

Fig. 16 All sizes in 12 map read and write speed. **a** Read, **b** write

Apache Spark is utilized to compute the clustering methods efficiently and then some comparison results are given.

3.3.1 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based data clustering algorithm. DBSCAN requires two parameters: (eps) and the minimum number of points required to form a dense region (minPts). It starts with an arbitrary starting point that has not been visited. This point's -neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized -environment of a different



(a)

	1	2	3	4	5	6	7	8	9	10	11	12
12GB/Avg1G	2.223	4.069	4.919	6.339	7.179	8.184	9.187	9.961	10.837	11.279	12.059	12.217
60GB/Avg1G	1.466	2.257	3.350	4.541	5.581	6.369	7.158	8.148	8.918	9.626	10.365	11.185
120GB/Avg1G	1.070	1.964	3.183	4.204	5.257	6.054	7.098	7.802	8.556	9.237	9.971	10.670

(b)

	1	2	3	4	5	6	7	8	9	10	11	12
12GB/Avg1G	3.56492	3.94325	2.96808	3.31758	3.06383	2.98433	2.98533	2.98458	2.666	2.47833	2.74	2.56475
60GB/Avg1G	2.00603	1.95533	1.78603	1.7535	1.61017	1.40678	1.50842	1.40247	1.35308	1.00193	1.18655	1.11922
120GB/Avg1G	1.71371	1.47357	1.81644	1.60672	1.40561	1.47347	1.41452	1.25569	0.83675	1.03818	1.03702	0.98673

Fig. 17 All sizes in 12 map read and write average speed of 1GB. **a** Read, **b** write

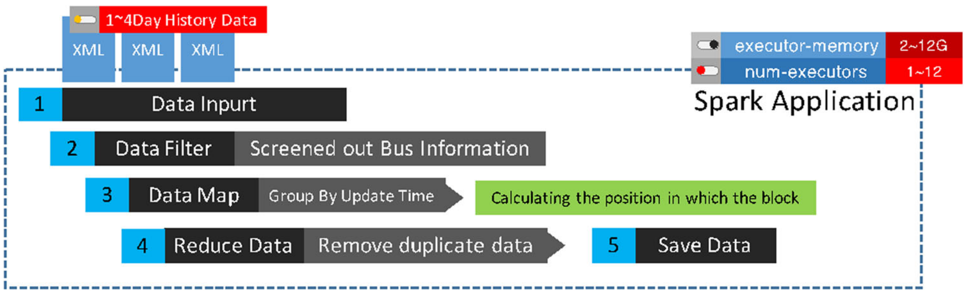


Fig. 18 The flow chart of convBus

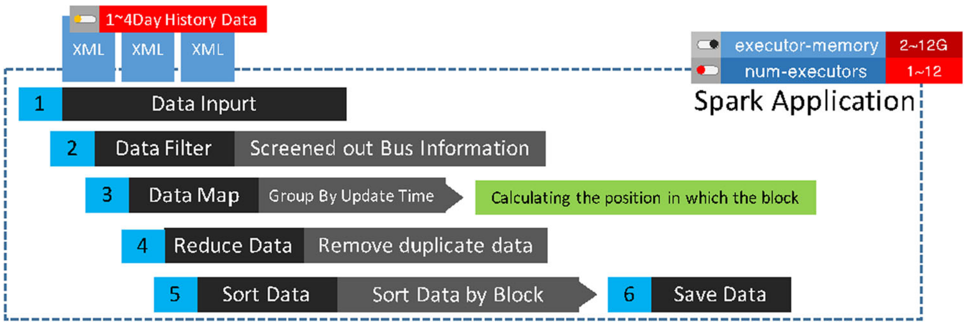


Fig. 19 The flow chart of comBus

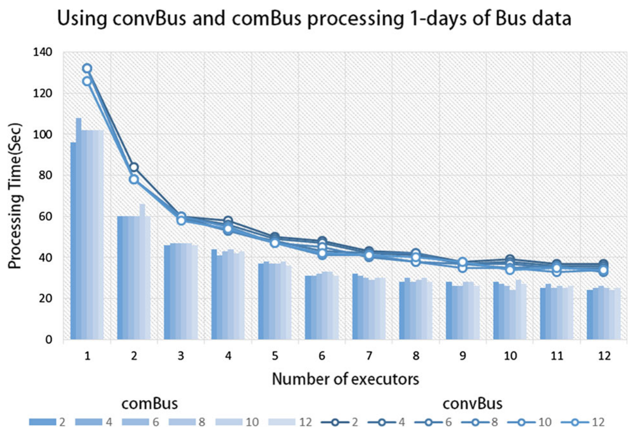


Fig. 20 Using convBus and comBus processing 1-days of Bus data

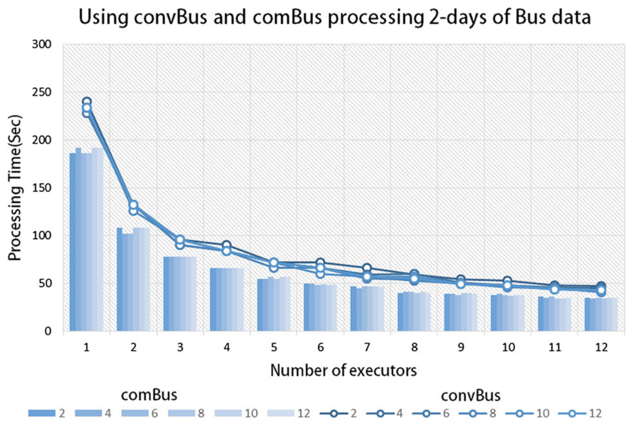


Fig. 21 Using convBus and comBus processing 2-days of Bus data

point and hence be made part of a cluster. If a point is found to be a dense part of a cluster, its -neighborhood is also part of that cluster. Hence, all points that are found within the -neighborhood are added, as is their own -neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise [20]. We utilize two epsilons and

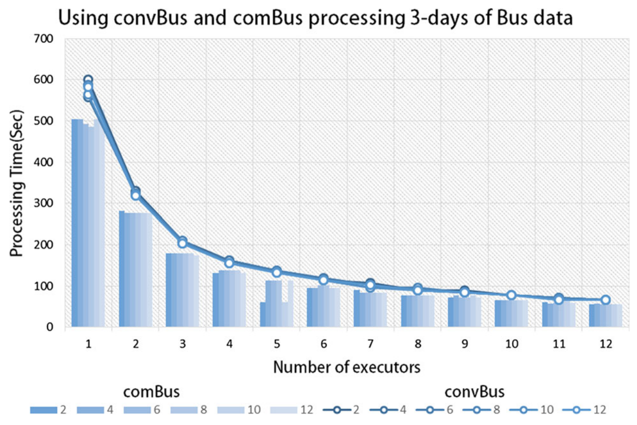


Fig. 22 Using convBus and comBus processing 3-days of Bus data

three minPts to test the performance of DBSCAN clustering. Figure 23(a–d) show the results in cases original, PTS = 3, PTS = 4, PTS = 5 under epsilon = 0.001. Figure 24(a–d) show the results in cases original, PTS = 3, PTS = 4, PTS = 5 under epsilon = 0.002. One can observe that the cluster decreases when PTS increases and the searching area enlarges when epsilon increases.

3.3.2 K-means

K-Means clustering is a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Given a set of observations $(x_1, x_2, \ldots, x_n)$ where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets $\{s_1, s_2, \ldots, s_k\}$ so as to minimize the within-cluster sum of squares

$$\arg min \sum_{i=1}^{k} \sum_{x_j \in s_i} \left\| x_j - \mu_i \right\|^2,$$

where μ_i is the mean of points in s_i. [21,22] We utilize k = 3 and two iterations to test the performance of K-MEANS clustering. Figure 25(a–d) show the results in cases original, k = 10, k = 50, k = 100 under iteration = 100. Figure 26(a–d) show the results in cases original, k = 10, k = 50, k = 100 under iteration = 1000. One can observe that the clustering fails when k is very small and works when k is sufficient.

3.3.3 Fuzzy C-means

The fuzzy C-means (FCM) [23,24] attempts to partition a finite collection of n elements

$$X = \{x_1, \ldots, x_n\}$$

into a collection of c fuzzy clusters with respect to some given criterion. Given a finite set of data, the algorithm returns a list of p cluster centres

$$C = \{c_1, \ldots, c_p\}$$

and a partition matrix

$$U = \left[u_{ij}\right]_{p\times n}$$

where each element u_{ij} tells the degree to which element x_j belongs to cluster c_i. Then FCM aims to minimize an objective function J:

$$J(U, c_1, c_2, \ldots, c_p) = \sum_{i=1}^{p} \sum_{j=1}^{n} (u_{ij})^m dist(c_i, x_j)^2$$

where

$$m \in [1, \infty)$$

represent weighting;

$$dist(c_i, x_j)$$

denotes the distance between c_i and x_j. By introducing Lagrange multiplier λ_i, the above objective function is rewritten as

$$\begin{aligned} &J_{new}\left(U, c_1, c_2, \ldots, c_p, \lambda_1, \ldots, \lambda_n\right) \\ &\quad= J(U, c_1, c_2, \ldots, c_p) + \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{p} u_{ij} - 1\right) \\ &\quad= \sum_{i=1}^{p} \sum_{j=1}^{n} (u_{ij})^m dist(c_i, x_j)^2 + \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{p} u_{ij} - 1\right) \end{aligned}$$

and the optimal cluster is

$$c_i = \frac{\sum_{j=1}^{n} (u_{ij})^m x_j}{\sum_{j=1}^{n} (u_{ij})^m} \tag{4}$$

We utilize three k-values and two iterations to test the performance of K-MEANS clustering. Figure 27(a–d) show the results in cases original, k = 10, k = 50, k = 100 under iteration = 50. Figure 28(a–d) show the results in cases original, k = 10, k = 50, k = 100 under iteration=100. One can observe that the clustering fails when k is very small and works when k is sufficient.

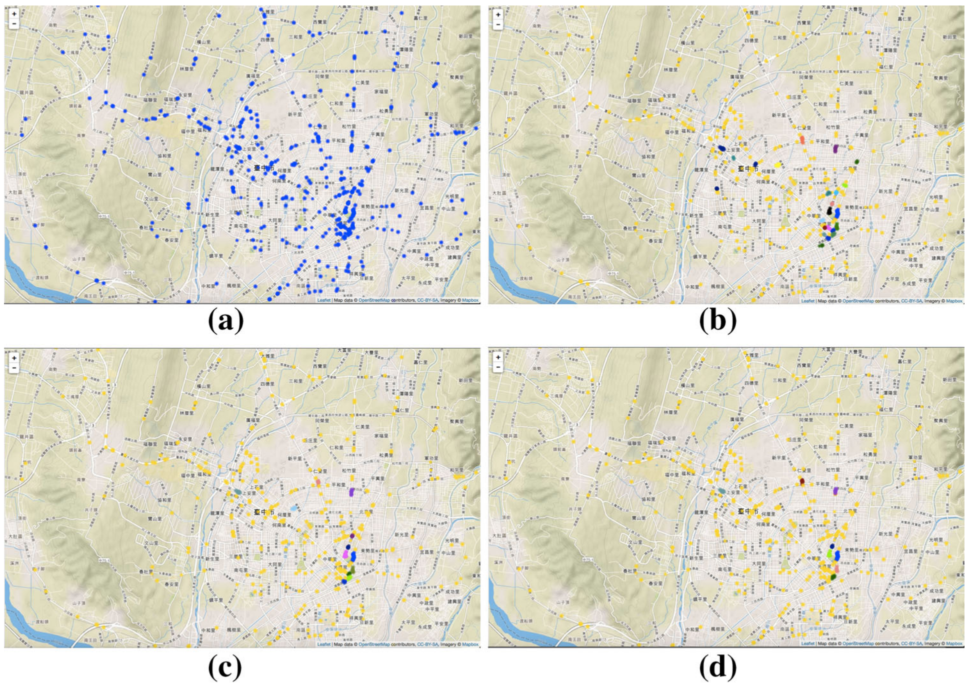


Fig. 23 DBSCAN in epsilon = 0.001 clustering results. **a** Original data, **b** EPS = 0.001/PTS = 3, **c** EPS = 0.001/PTS = 4, **d** EPS = 0.001/PTS = 5

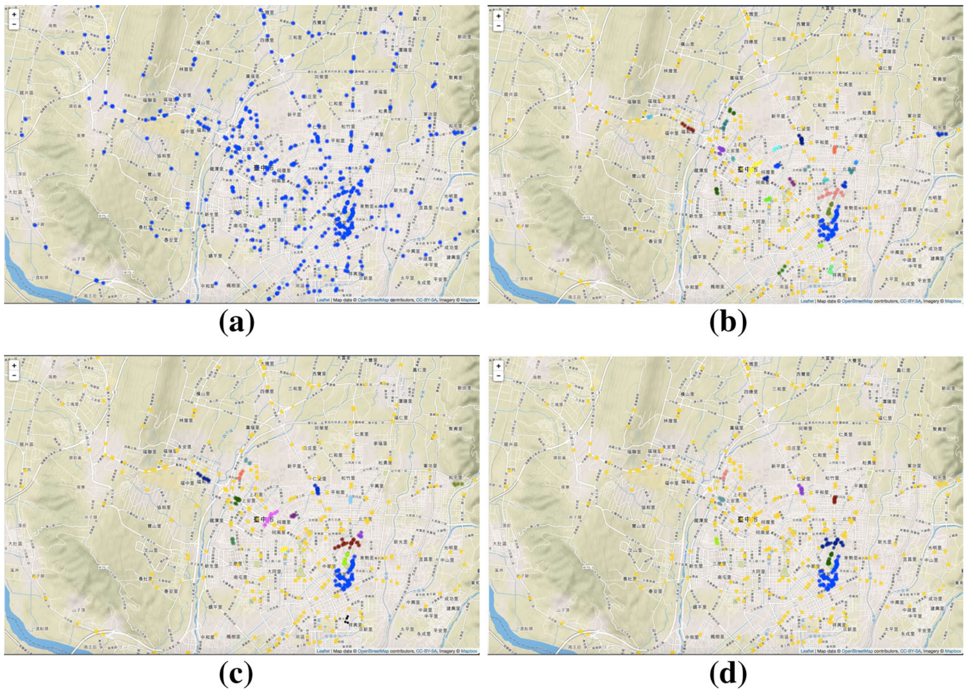


Fig. 24 DBSCAN in epsilon = 0.002 clustering results. **a** Original data, **b** EPS = 0.001/PTS = 3, **c** EPS = 0.001/PTS = 4, **d** EPS = 0.001/PTS = 5

3.4 Results comparison

Figure 29(a–d) show the clustering results in cases original, DBSCAN with EPS = 0.001/PTS = 5, K-means with k-value = 100/Iteration = 100, Fuzzy-C-means with k-value = 100/Iterations = 50. From the results in Figure 5, we know that Fuzzy-C-Means has the best clustering but k-means and DBSCAN use less time.

4 Experimental environment and results

In this section, experimental environment and results with respect to the proposed Cloud City Traffic State Assessment System using open data, GPS, GPRS in Taichung city, Taiwan are present. Figure 30 shows Taichung City bus Dynamic Positioning. We adopt K-Means clustering to efficiently find the area of traffic jam in Taichung city, Taiwan and then the

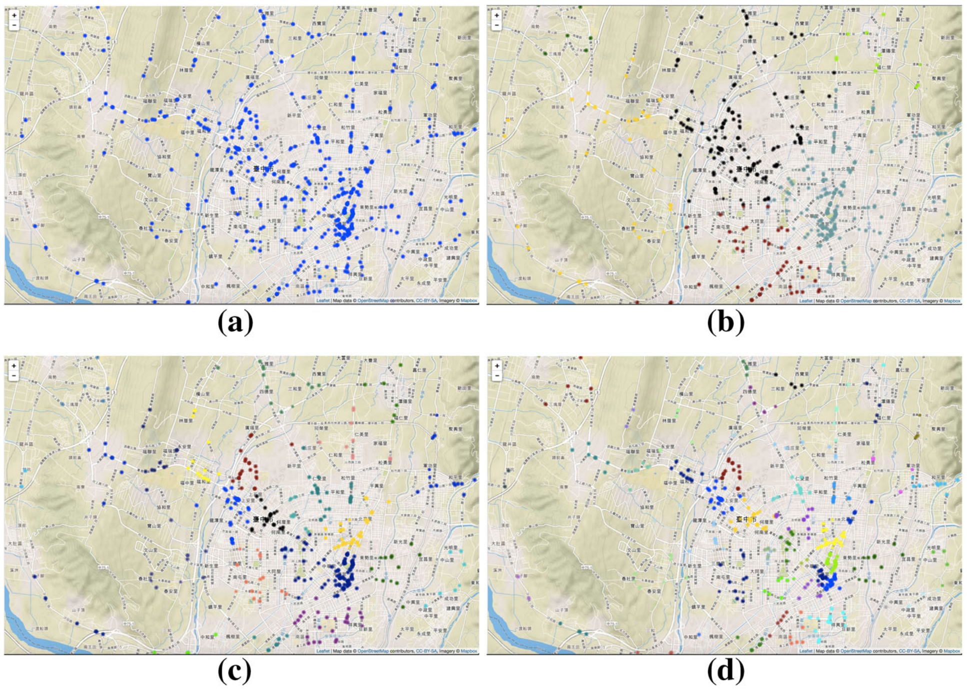


Fig. 25 K-MEANS in iterations = 100 clustering results. **a** Original data, **b** clusters = 10/iterations = 100, **c** clusters = 50/iterations = 100, **d** clusters = 100/iterations = 100

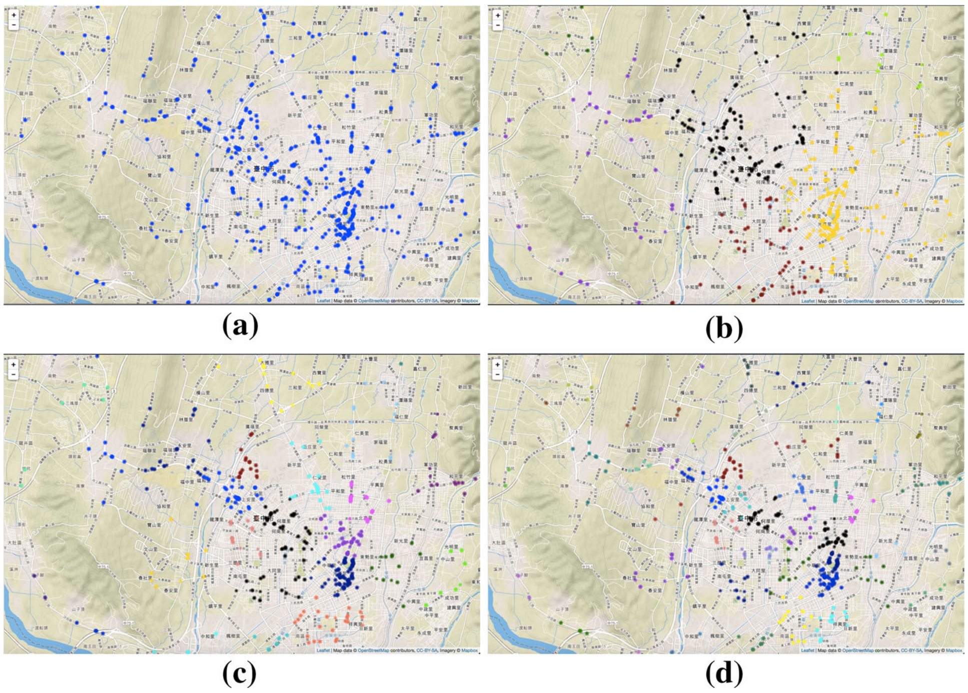


Fig. 26 K-MEANS in iterations = 1000 clustering results. **a** Original data, **b** clusters = 10/Iterations = 100, **c** clusters = 50/Iterations = 100, **d** clusters = 100/iterations = 100

proposed irregular moving average is utilized to find the area of traffic jam in Taiwan Boulevard which is the main road in Taichung city. Based on these experimental results, the provided system services are present via an advanced web technology.

4.1 Experimental environment

This subsection introduces our environmental environment including hardware and software. To implement the proposed system, we use 12 physical servers connected by Giga-

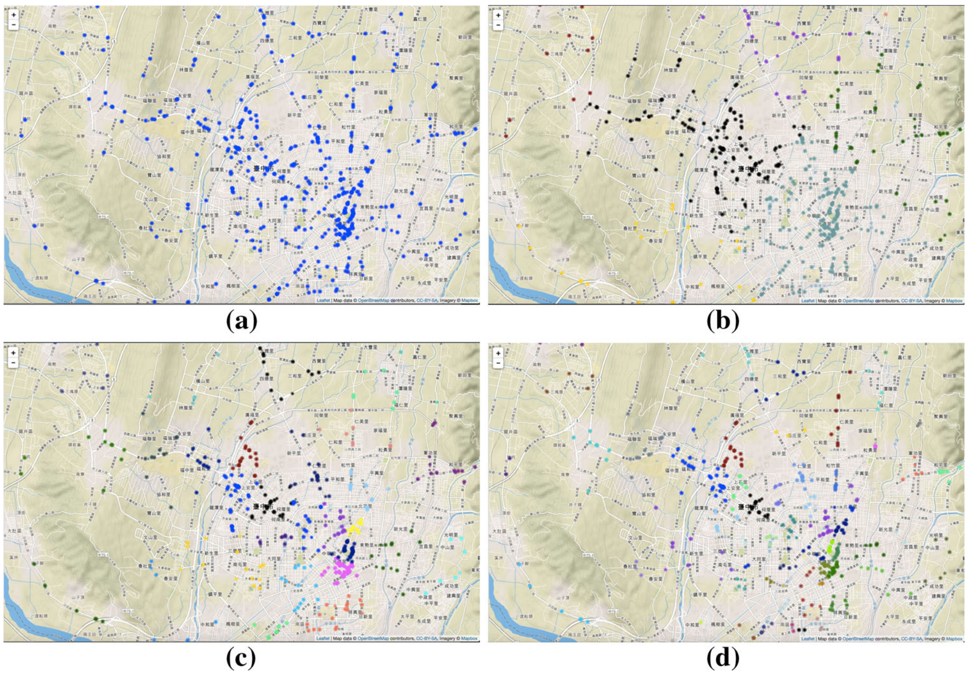


Fig. 27 Fuzzy-C-means in iterations = 50 Clustering results. **a** Original data, **b** clusters = 10/Iterations = 50, **c** clusters = 50/Iterations = 50, **d** clusters = 100/Iterations = 50

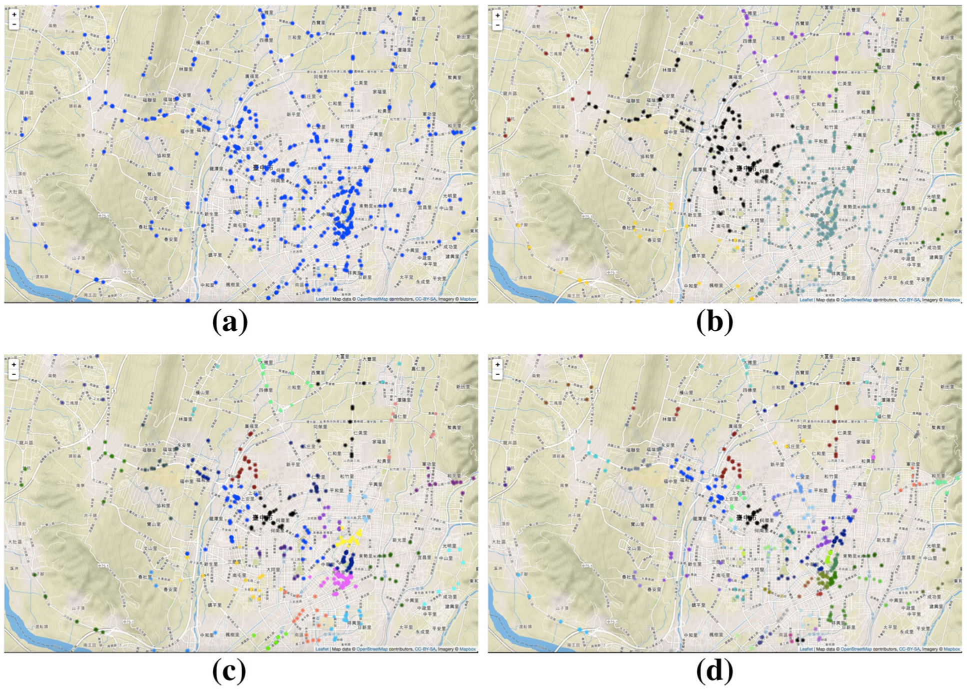


Fig. 28 Fuzzy-C-means in iterations = 100 Clustering results. **a** Original data, **b** clusters = 10/Iterations = 100, **c** clusters = 50/Iterations = 100, **d** clusters = 100/Iterations = 100

bit Ethernet to establish a cluster as shown in Fig. 31. In hardware, each physical server is Intel Core i7 CPU with 16GB Memory and 1TB HD. In software, Ubuntu 14.04 is adopted as our operating systems. Also, Cloudera Express 5.2.0 Hadoop 2.5.0 HBase 0.98.6 Spark 1.1.0 Zookeeper 3.4.5 are installed.

4.2 Cloud city traffic state assessment system

In this work, the proposed Cloud City Traffic State Assessment System offers the user to understand the Traffic State through a Web-based User-friendly interface using Html5, CSS3, JavaScript, and JQuery with semantic Front End. Fig-

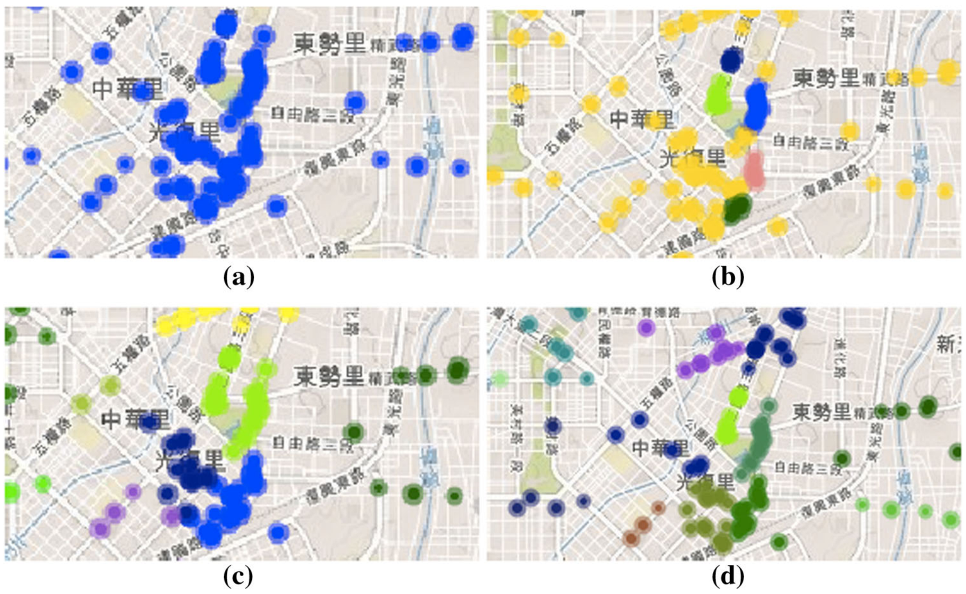


Fig. 29 Comparison of DBSCAN,K-means, fuzzy-C-means. **a** Original data, **b** DBSCAN EPS = 0.001/PTS = 5, **c** K-means clusters = 100/Iterations = 100, **d** fuzzy-C-means clusters = 100/Iterations = 50

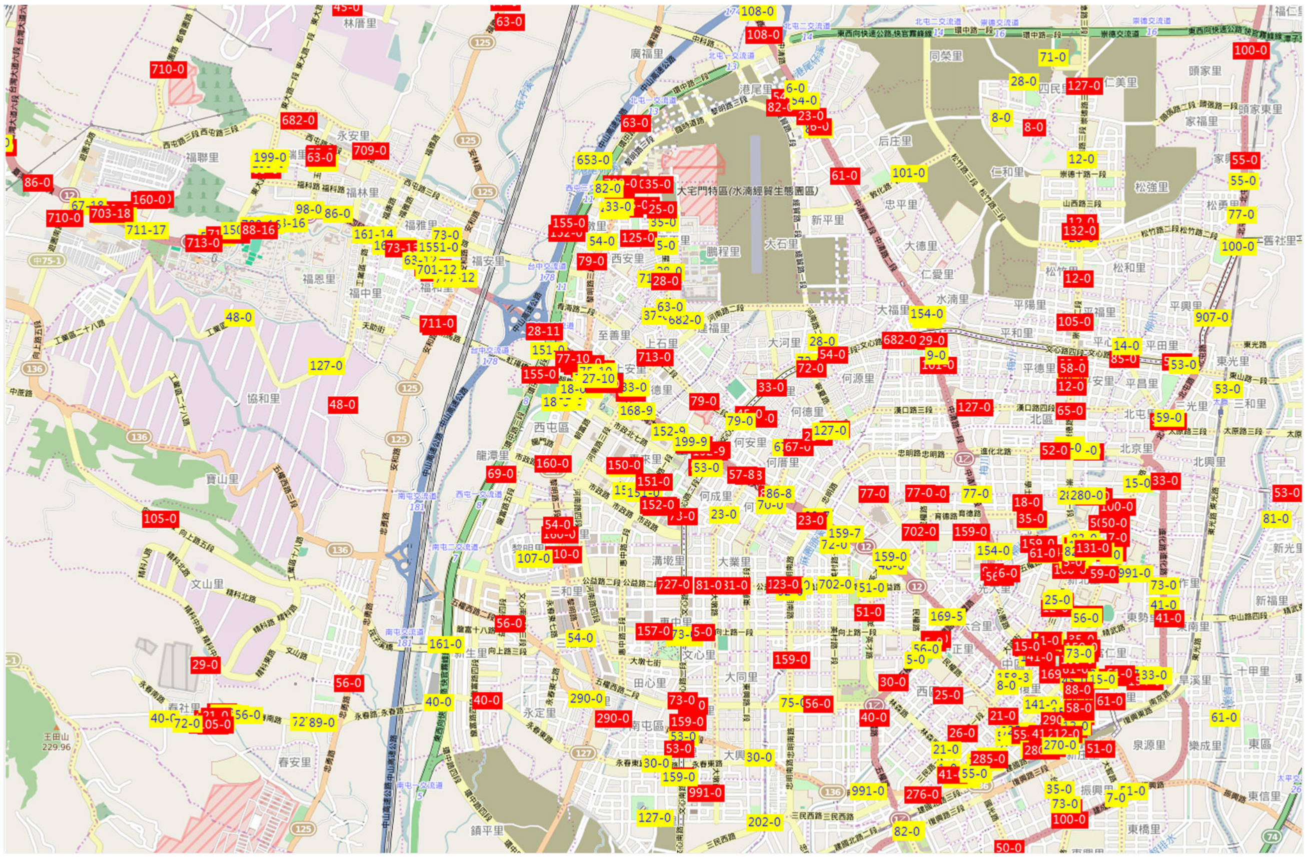


Fig. 30 Taichung City bus dynamic positioning

Fig. 31 Spark and Hadoop computing cluster

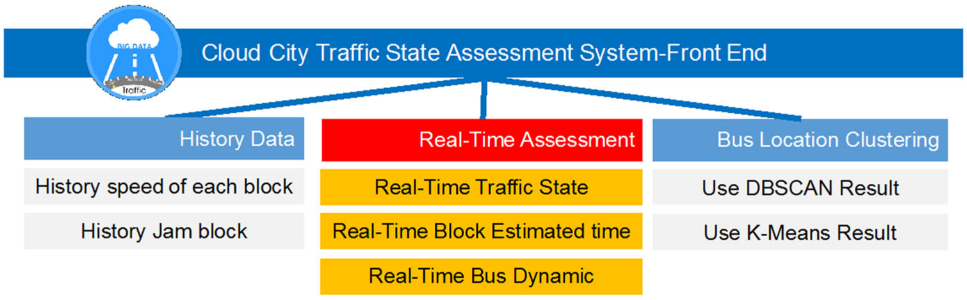
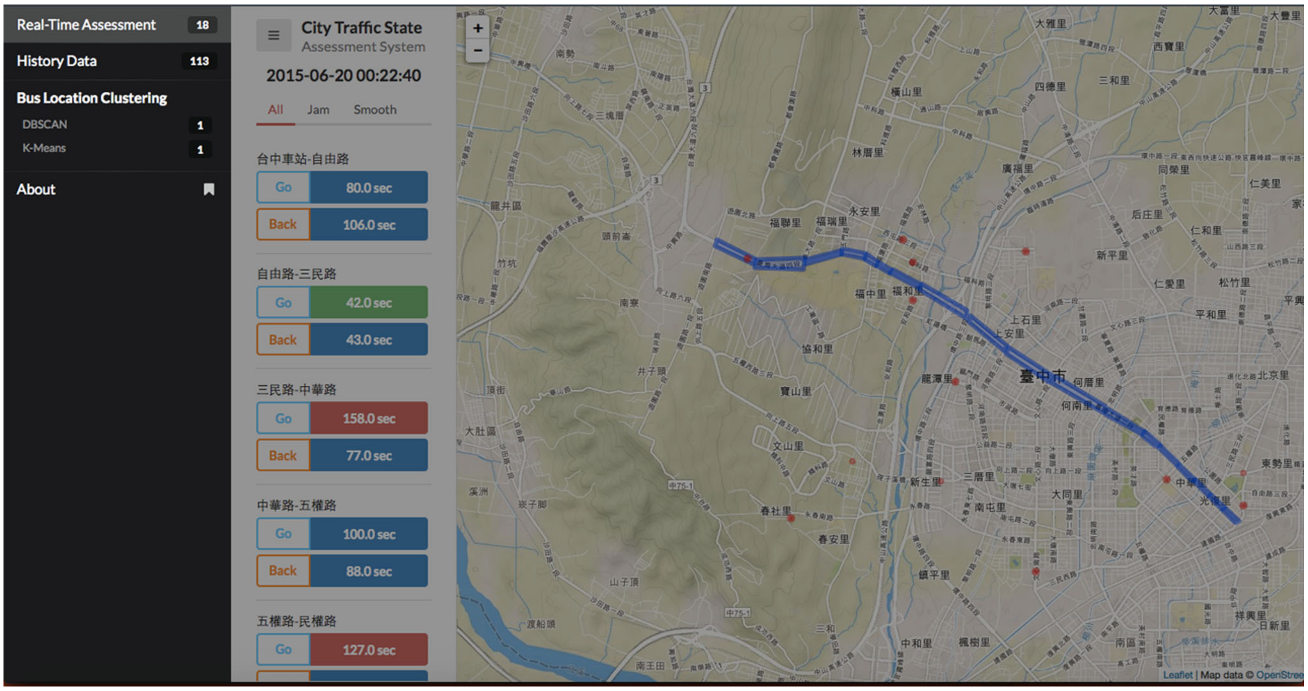


Fig. 32 Cloud city traffic state assessment system functions



2015-06-20 00:22:40

台中車站-自由路
Go 80.0 sec
Back 106.0 sec

自由路-三民路
Go 42.0 sec
Back 43.0 sec

三民路-中華路
Go 158.0 sec
Back 77.0 sec

中華路-五權路
Go 100.0 sec
Back 88.0 sec

五權路-民權路
Go 127.0 sec

Fig. 33 Web UI function menu

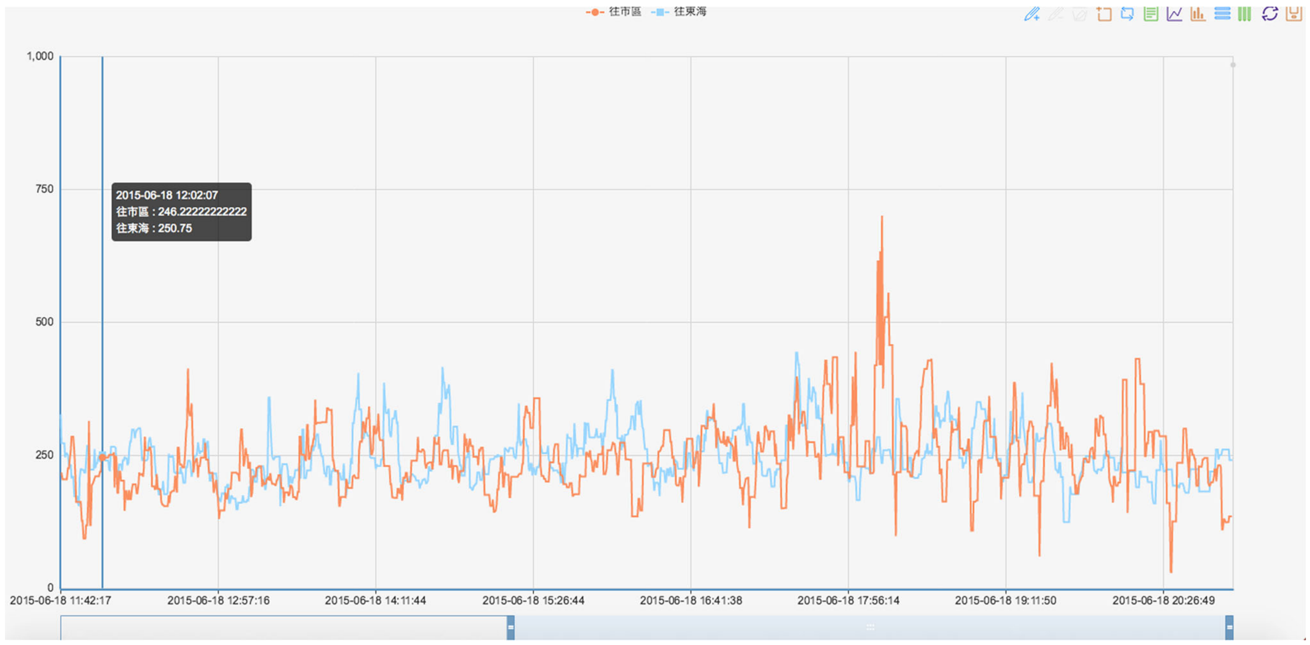
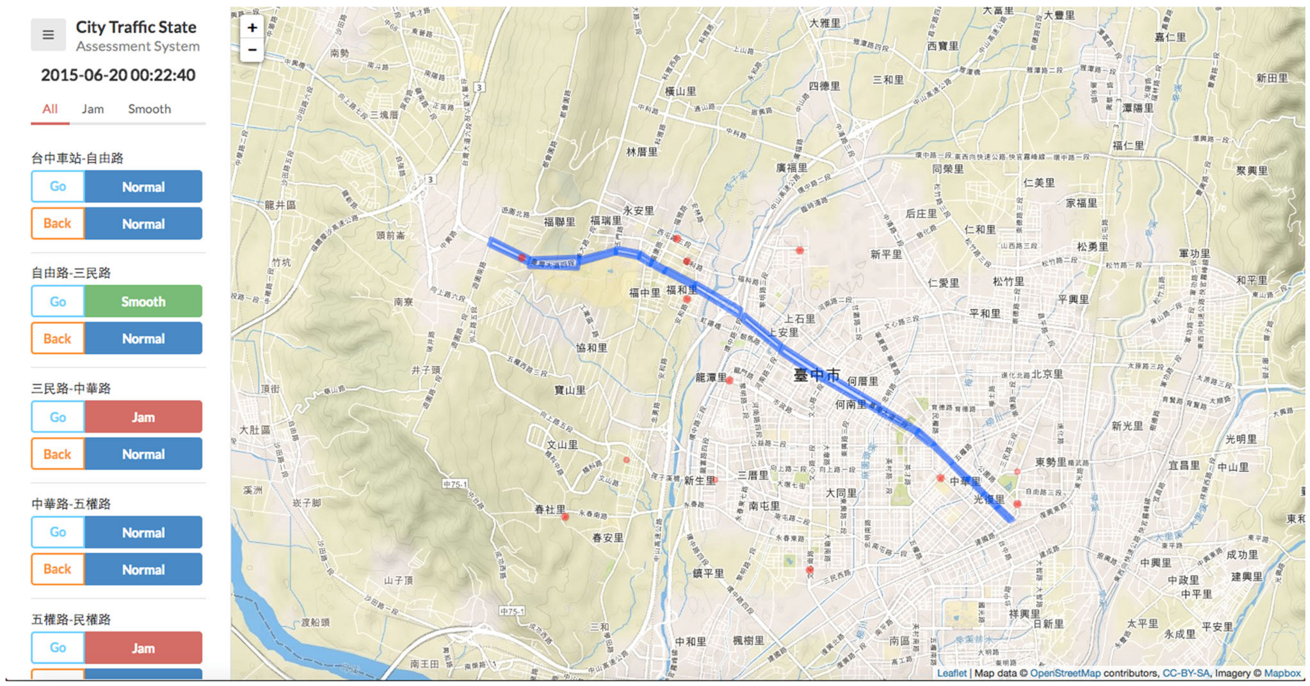


Fig. 34 History bus travel each block of time use waveform display


City Traffic State
Assessment System
2015-06-20 00:22:40
All Jam Smooth

台中車站-自由路
Go Normal
Back Normal

自由路-三民路
Go Smooth
Back Normal

三民路-中華路
Go Jam
Back Normal

中華路-五權路
Go Normal
Back Normal

五權路-民權路
Go Jam

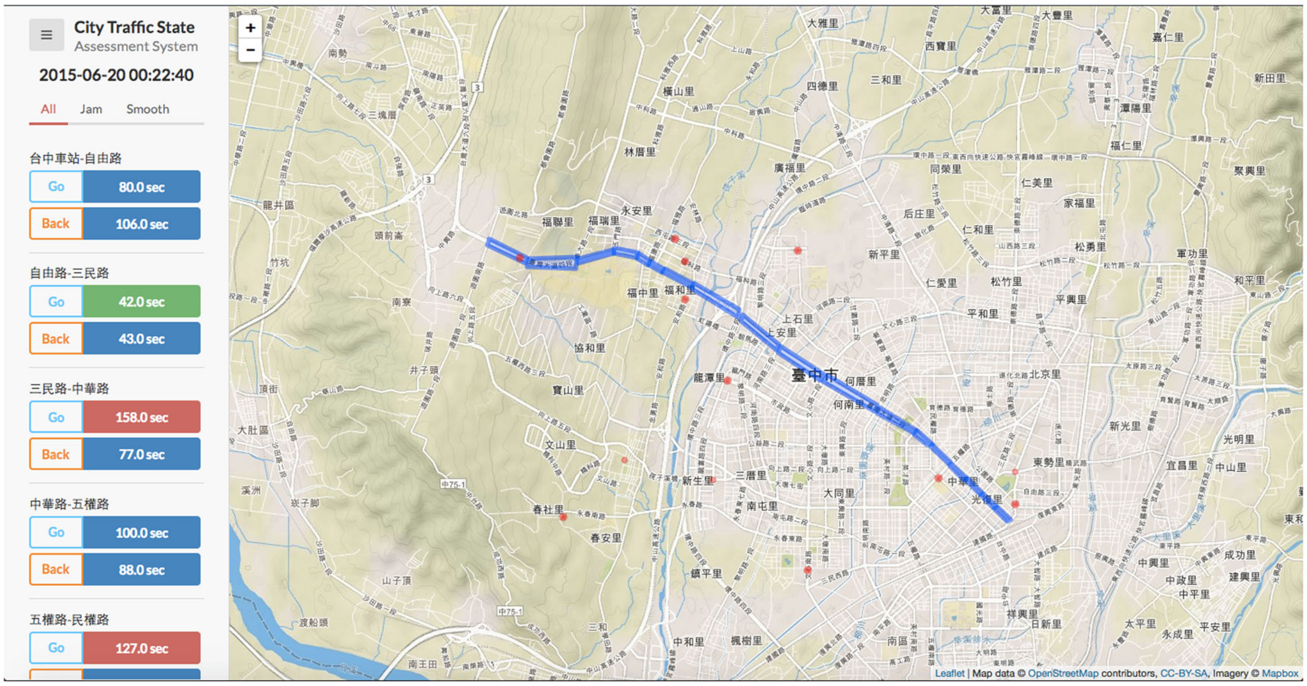
臺中市
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Fig. 35 Real-time assessment traffic in Taiwan Boulevard using WEB presentation

ure 32 and the left side of Fig. 33 show the following three results.

– *Historical data* provide the average speed of a bus and the area of traffic jam in the past by a line chart, as shown in Fig. 34.
– *Real-time evaluation* provide real-time traffic state, bus speed, and distribution of busses, as shown in Figs. 35,36.
– *Clustering results* provide the clustering results of DBSCAN, K-Means and Fuzzy C-means.

Moreover, we use AJAX and Openstreetmap to update webpage information and Map information.


City Traffic State
Assessment System

2015-06-20 00:22:40

All Jam Smooth

台中車站-自由路

Go	80.0 sec
Back	106.0 sec

自由路-三民路

Go	42.0 sec
Back	43.0 sec

三民路-中華路

Go	158.0 sec
Back	77.0 sec

中華路-五權路

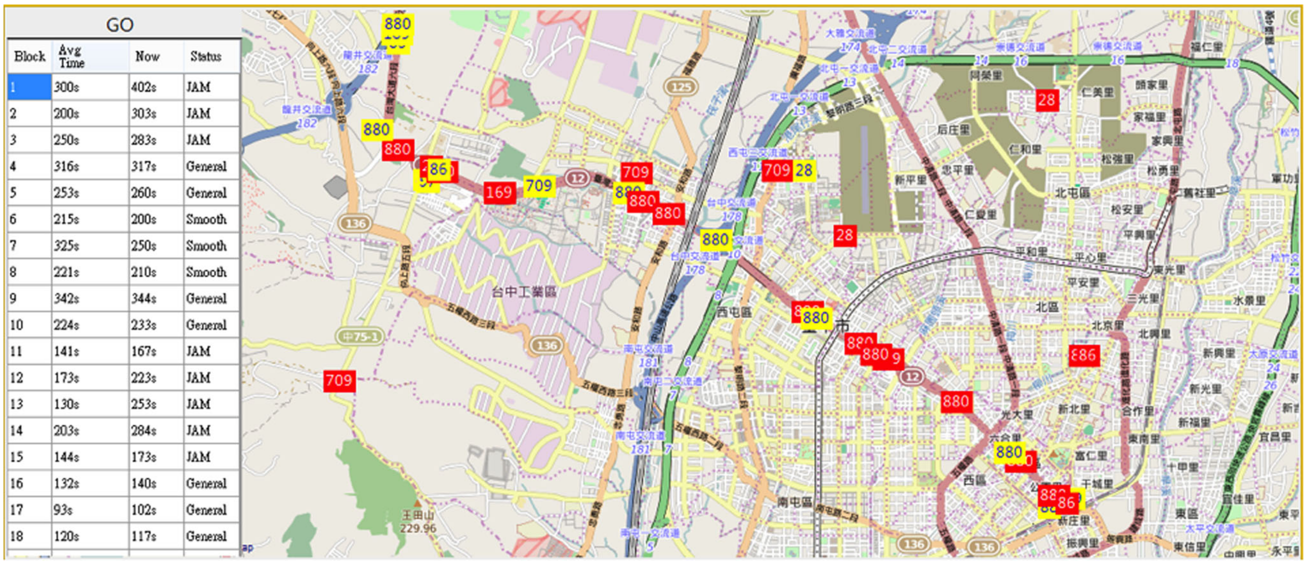
Go	100.0 sec
Back	88.0 sec

五權路-民權路

Go	127.0 sec

臺中市
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Fig. 36 Real-time assessment traffic in Taiwan Boulevard using WEB presentation


GO

Block	Avg Time	Now	Status
1	300s	402s	JAM
2	200s	303s	JAM
3	250s	283s	JAM
4	316s	317s	General
5	253s	260s	General
6	215s	200s	Smooth
7	325s	250s	Smooth
8	221s	210s	Smooth
9	342s	344s	General
10	224s	233s	General
11	141s	167s	JAM
12	173s	223s	JAM
13	130s	253s	JAM
14	203s	284s	JAM
15	144s	173s	JAM
16	132s	140s	General
17	93s	102s	General
18	120s	117s	General

Fig. 37 Cloud city traffic state assessment system in Taiwan Boulevard

4.3 Cloud city traffic state assessment system in Taiwan Boulevard

This subsection implements the proposed Cloud City Traffic State Assessment System in Taiwan Boulevard. First, the host road is randomly divide into several blocks according to the intersection with other roads is shown in Figs. 8 and 10 and k-means clustering is applied to roughly classify the blocks and their size in Taichung city, Taiwan. Next, the real-time buses average velocity is compared with the past average velocity in each block to indicate whether the traffic in each block is heavy by using irregular moving average. As shown in resulting table of Fig. 37, the second column shows the past average velocity while the third column shows the real-time buses average velocity. The forth column shows the traffic state in each block.

5 Conclusions and future work

In this work, we present a cloud city traffic state assessment system using a novel architecture of big data. With the high-scalability cloud technologies, Hadoop and Spark, the proposed system architecture for big data and services is implemented in Taiwan Boulevard efficiently. In addition, experimental results show that Spark adopted in our system has better performance than Hadoop MapReduce. For the system interface, this work designed a front-end web interface providing users to view traffic status in Taiwan Boulevard so that a user can both know the real-time traffic state and view the history of traffic status. In the future, we expect to apply this system to all roads in Taichung and to improve the accuracy of traffic state assessment.

Acknowledgements This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grants numbers 104-2221-E-029-010-MY3, 103-2632-H-029-001-MY2, and 105-2634-E-029-001.

References

1. Wang, L., Lu, K., Liu, P., Ranjan, R., Chen, L.: IK-SVD: dictionary learning for spatial big data via incremental atom update. Comput. Sci. Eng. **16**, 41–52 (2015a)
2. Wang, L., Geng, H., Liu, P., Lu, K., Kolodziej, J., Ranjan, R., Zomaya, A.Y.: Particle swarm optimization based dictionary learning for remote sensing big data. Knowl. Based Syst. **79**, 43–50 (2015b)
3. Ma, Y., Wang, L., Liu, P., Ranjan, R.: Towards building a data-intensive index for big data computing—a case study of remote sensing data processing. Inf. Sci. **319**, 171–188 (2015)
4. Ibm smarter business. http://public.dhe.ibm.com/software/uk/itsolutions/businessconnect2013/dk_pdf/SmarterBusDenmark_BigData_MDoylev2external.pdf (2013)
5. Barbierato, E., Gribaudo, M., Iacono, L.: Performance evaluation of nosql big-data applications using multi-formalism models. Future Gener. Comput. Syst. **37**, 345–353 (2014)
6. Zhang, C., Liu, X.: Hbasemq: a distributed message queuing system on clouds with hbase. In: Proceedings IEEE, INFOCOM, pp. 40–44 (2013)
7. Yang, C.T., Liao, C.J., Liu, J.C., Den, W., Chou, Y.C., Tsai, J.J.: Construction and application of an intelligent air quality monitoring system for healthcare environment. J. Med. Syst. **38**, 15 (2014)
8. Yang, C.-T., Shih, W.-C., Chen, L.-T., Kuo, C.-T., Jiang, F.-C., Leu, F.-Y.: Accessing medical image file with co-allocation hdfs in cloud. Future Gener. Comput. Syst. **43–44**, 61–73 (2015)
9. Gu, L., Li, H.: Memory or time: Performance evaluation for iterative operation on hadoop and spark. In: IEEE 10th International Conference on, High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), pp. 721–727 (2013)
10. Urbani, J., Margara, A., Jacobs, C., Voulgaris, S., Bal, H.: Ajira: A lightweight distributed middleware for mapreduce and stream processing. In: IEEE 34th International Conference on, Distributed Computing Systems (ICDCS), pp. 545–554 (2014)
11. Zhang, J., You, S., Gruenwald, L.: High-performance spatial query processing on big taxi trip data using gpgpus. In: IEEE International Congress on, Big Data (BigData Congress), pp. 72–79 (2014)
12. Wang, L., Hu, S.W., Liu, P.: A computing perspective on smart city. IEEE Trans. Comput. **65**, 1337–1338 (2016)
13. Dobre, C., Xhafa, F.: Intelligent services for big data science. Future Gener. Comput. Syst. **37**, 267–281 (2014)
14. Zeng, Y., Lan, J., Ran, B., Jiang, Y.: A novel multisensor traffic state assessment system based on incomplete data. ScientificWorld J. **2014**, 532602 (2014)
15. Jin, Y., Deyu, T., Yi, Z.: A distributed storage model for ehr based on hbase. In: International Conference on, Information Management, Innovation Management and Industrial Engineering (ICIII), vol. 2, pp. 369–372 (2011)
16. Ding, H., Jin, Y., Cui, Y., Yang, T.; Distributed storage of network measurement data on hbase. In: IEEE 2nd International Conference on, Cloud Computing and Intelligent Systems (CCIS), vol. 02, pp. 716–720 (2012)
17. Vora, M.: Hadoop-hbase for large-scale data. In: International Conference on, Computer Science and Network Technology (ICCSNT), vol. 1, pp. 601–605 (2011)
18. The hadoop distributed file system: Architecture and design. http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf (2007)
19. Apache spark. https://spark.apache.org/ (2015)
20. Campello, R., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Pei, J., Tseng, V., Cao, L., Motoda, H., Xu, G. (eds.) Advances in Knowledge Discovery and Data Mining. Lecture Notes in Computer Science, vol. 7819, pp. 160–172. Springer, Berlin (2013)
21. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability Statistics, vol. 1, pp. 281–297. University of California Press, Berkeley, (1967)
22. Frahling, G., Sohler, C.: A fast k-means implementation using coresets. In: Proceedings of the 22nd ACM Symposium on Computational Geometry, pp. 135–143. Sedona, Arizona, USA, June 5–7, (2006)
23. Fuzzy clustering. http://en.wikipedia.org/wiki/Fuzzy_clustering (2015)
24. Scala of fuzzy-c-means clustering. http://gist.github.com/kralo/8721440 (2015)

Chao-Tung Yang is a Professor of Computer Science at Tunghai University in Taiwan. He received the Ph.D. in Computer Science from National Chiao Tung University in July 1996. In August 2001, he joined the Faculty of the Department of Computer Science at Tunghai University. He is serving in a number of journal editorial boards, including International Journal of Communication Systems, Journal of Cloud Computing, "Grid Computing, Applications and Technology" Special Issue of Journal of Supercomputing, and "Grid and Cloud Computing" Special Issue of International Journal of Ad Hoc and Ubiquitous Computing. Dr. Yang has published more than 200 papers in journals, book chapters and conference proceedings. His present research interests are in cloud computing and service, grid computing, parallel computing, and multicore programming. He is a member of the IEEE Computer Society and ACM.

Shuo-Tsung Chen received the B.S. degree in Mathematics from National Cheng Kung University, Tainan in 1996 and M.S. degree in Applied Mathematics from Tunghai University, Taichung in 2003, Taiwan. In 2010, he received the Ph.D. degree in Electrical Engineering from National Chinan University, Nantou, Taiwan. Now he is an Adjunct Assistant Professor in Department of Electronic Engineering, National Formosa University and Department of Applied Mathematics, Tunghai University.

Yin-Zhen Yan received the B.S. degree in Computer Science and Information Engineering from HungKuang University, Tainan in 2013 and M.S. degree in Computer Science from Tunghai University, Taichung in 2015, Taiwan. Now he is a computer engineer.