Air Quality Monitoring System and Benchmarking

Liu, Xiufeng; Nielsen, Per Sieverts

Published in:
International Conference on Big Data Analytics and Knowledge Discovery

Link to article, DOI:
10.1007/978-3-319-64283-3_34

Publication date:
2017

Air Quality Monitoring System and Benchmarking

Xiufeng Liu, and Per Sieverts Nielsen

Technical University of Denmark
{xiuli,pernn}@dtu.dk

Abstract. Air quality monitoring has become an integral part of smart city solutions. This paper presents an air quality monitoring system based on Internet of Things (IoT) technologies, and establishes a cloud-based platform to address the challenges related to IoT data management and processing capabilities, including data collection, storage, analysis, and visualization. In addition, this paper also benchmarks four state-of-the-art database systems to investigate the appropriate technologies for managing large-scale IoT datasets.

Keywords: IoT-based, Dashboard, Cloud computing, Benchmarking

1 Introduction

With the development of urbanization, cities are facing an increasing challenge to reduce the climate impact. Cities are the large energy consumers accounting for 80% of the total carbon emissions [10]. Carbon emissions are mainly responsible for the greenhouse effect, and close monitoring of carbon emissions is an effective way to reduce the climate impact. Today more than 1,400 cities worldwide regularly report on their greenhouse gas (GHG) emissions through the Carbon Climate Register and the Governors Convention initiative [1]. However, emissions monitoring at the city level is often costly and time-consuming because they relate to a high degree of uncertainty. Most cities in Europe do not currently possess the capacity to measure the actual emissions within their urban space. On the other hand, carbon reduction has become a city development strategy. For example, the European Union (EU) aims to cut its primary energy consumption by 27% by 2030. In Denmark, the government has set the goal of reducing GHG emission by 40% by 2020, and becoming a fossil-fuel free country by 2050. This requires innovating approaches to reporting air quality for politician and citizens to make quick and effective decision makings. In this context, IoT technologies can be used to address the challenge of real-time monitoring of air quality, such as the detection of pollutant concentration levels and trends.

In this paper, we present an IoT-based air quality monitoring system developed under our *Carbon Track and Trace (CTT)* project [3]. The system can track the real-time greenhouse gas emissions at the urban-street level. This work makes the following contributions: First, we present an IoT-based solution for air quality monitoring. Second, we propose a cloud-based platform for managing air quality and other IoT related datasets, which provides the necessary functionality for adding high-frequency sensor data and accessing data from the database. Third, we develop a dashboard for presenting the insight of the data. Fourth, we investigate and benchmark alternative database technolo-

gies for managing IoT datasets, including specialized time-series database, in-memory columnar store, key-value store and relational database.

The rest of this paper is organized as follows. Section 2 describes the system design and implementation. Section 3 benchmarks the state-of-the-art database systems for managing IoT data. Section 4 surveys the related works. Section 6 concludes the paper and present the future works.

2 System Design and Implementation

The system is designed with two purposes: one is to enable city officers, decision makers, citizens, and other stakeholders to visualize the emission measurements of the whole city for decision making or monitoring purpose, while the other is to establish a scalable data management system that can manage other IoT data used to study the impact on the air quality, such as weather data and traffic data. The data management system is required with a high scalability for supporting the large-scale deployments of sensors within the cities in the future. Besides, the platform should ease the integration of new data sources, and provide standard access service for enabling the data to be used by other applications or users. The system should make use of existing open source technologies, including the IoT network, data management, and visualization systems to reduce cost.

We develop the system based on the existing technologies, including sensors, sensor platforms, and IoT network. Air quality data (including CO_2 and NO_x) are collected by low-cost sensors deployed in outdoor environments. All sensor nodes are also equipped with climate sensors for collecting weather conditions (e.g., ambient temperature, pressure, wind speed and humidity) and particulate matter (PM) sensors for measuring dust particle size (e.g., PM1, 2.5 and 10). The sensors collect these measurements every five minutes and send them to the server on the cloud over Internet of Things network (TTN) via a low-power LoRaWAN gateway. The server aggregates the data from the TTN and saves them in the cloud database from which the dashboard reads the data to generate real-time views. Figure 1 shows the system architecture. In order to achieve high scalability, the architecture employs the distributed time series database, *OpenTSDB* [16], as the cloud database for managing air quality data, as well as weather and traffic data for correlation analysis. The reason for choosing OpenTSDB is that it is a NoSQL database that can handle massive amounts of data which is anticipated. As air quality data and their associated data (such as traffic and weather data) are captured, they will never be updated or changed. We expect thousands of sensors to be deployed in the city in the future. Therefore, it is preferable to select a distributed database to meet the city-scale IoT data management. OpenTSDB provides REST-based services for adding and accessing data, and supporting advanced queries for complex data analytics.

The time series of saving into OpenTSDB are given a unique name, called *metric*. To discriminate different data sources, we add a prefix to the name of metric, e.g., $AQ\_$ for air quality data, $TF\_$ for traffic data and $WT\_$ for weather data. In addition, each time series is labeled with multiple *tags* for its dimensions or features. A tag is a key/value pair, and multiple tags can be combined to for doing complex queries. List 1 is an example of a data point with a CO_2 metric tagged with the device id (`dev_eui`) and the sensor location (`longitude` and `latitude`).

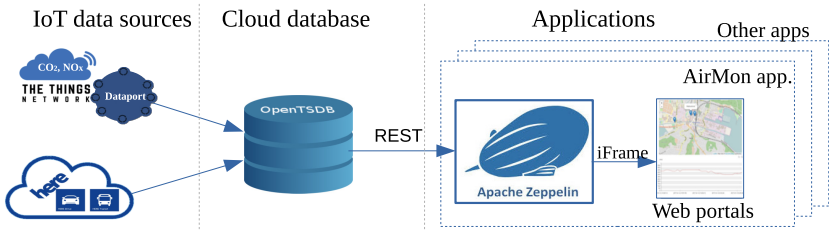

Fig. 1. Overview of the system architecture

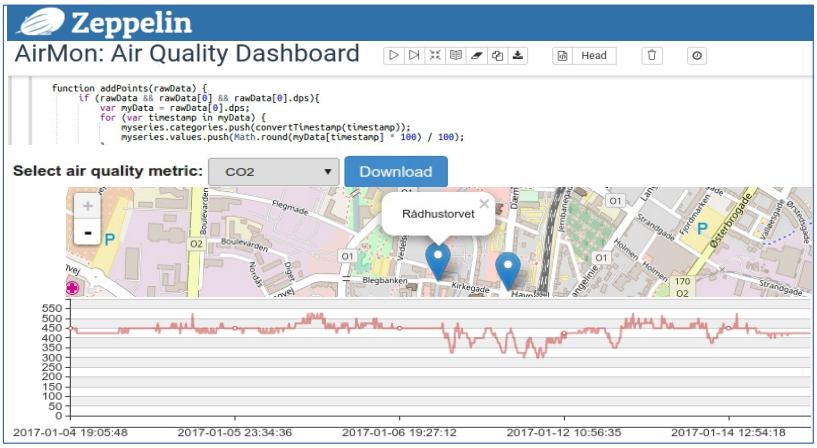

Fig. 2. Data analysis and visualization on Zeppelin

List 1: A single data point written to Opentsdb

```
{
  "metric": "AQ_GP_CO2",
  "timestamp": 1488207776,
  "value": 11.487,
  "tags": {
          "dev_eui": "00000000902FBDD2",
          "longitude": 9.5324,
          "latitude": 55.70805
  }
}
```

We use Apache Zeppelin as the visualization platform to implement the dashboard (see Figure 2). JavaScript is the programming language for implementing the visualization program, which accesses the data from OpenTSDB, and generates the real-time dashboard. The program is run on Zeppelin's Web-based interface (see the top in Figure 2). The open source Leaflet map visualizes the deployment locations of the sensors (see the middle in Figure 2). The real-time air quality time series will be displayed on the bottom chart if a sensor location marker is clicked (see the bottom in Figure 2). The chart is implemented using Highchart JavaScript library. The chart is updated dynamically when a new reading has been received. The chart can also be exported as an *iframe* to be embedded in other web pages, e.g., on a city government portal.

3 Benchmarking

3.1 Experimental settings

In this section, we will benchmark four representative database systems for managing IoT data, including OpenTSDB [16], BerkeleyDB [15], PostgreSQL [21] and KDB+ [11]. They are the state-of-the-art database technologies in the following four categories: distributed time series database (OpenTSDB), key-value store (BerkelyeDB), relational database (PostgreSQL), and column store (KDB+). BerkelyeDB and KDB+ are main memory based. KDB+ is the commercial database system, and we are permitted to publish its benchmarking data. Only OpenTSDB is an distributed database system. To align the settings, we evaluate OpenTSDB on a single server environment.

We benchmark the database systems in the private Cloud (*SciCloud, www.science-cloud.dk*). A virtual machine instance is created, with 8 cores and 16GB RAM, running 64-bit Ubuntu 16.04. PostgreSQL 9.6 is used, with the settings "shared buffers = 4096MB, temp buffers = 512MB, work mem = 1024MB, checkpoint segments = 64" and default values for the other configuration parameters. KDB+ and BerkeleyDB use their default settings.

The test data are the messages streamed from sensors with 10 metrics. For n sensors, the total number of time series (metrics) is $10 \times n$. At present, since we have only 14 sensors, we use a data generator to generate more datasets, which simply replace the numeric and string typed attribute values with random numbers. These values are not important because we are only benchmarking the performance of the database systems.

3.2 Benchmarking methods

The purpose of the benchmarking is to evaluate the ability of the database systems (i) to add incoming IoT data with a high speed, (ii) to store big IoT data over a long period of the time, and (iii) to provide data as a service. We use the appropriate data model supported by each database system for the benchmarking. This means that OpenTSDB uses its default data model; BerkelyDB uses key-value pair data model, where device id and metric id are saved as the key, bundled with the timestamp as a key object, and the other attributes are saved as the value; PostgreSQL uses a relational data model, with the table layout of *(device id, metric, metric value, timestamp, tags)* with a composite index created on the device id and metric attribute; KDB+ uses a columnar data model with its default settings.

The pseudocode for loading and querying is shown in Algorithm 1 and 2, respectively. The target table is empty before loading. We insert the required number of data points into the table, measure the elapsed time and calculate the throughput for every 10,000 data points added. This approach allows us to investigate the impact of the amount of data on the subsequent load performance. The target table is padded with the data before the query is benchmarked. We are interested in evaluating the queries that need to scan the table, such as aggregations of min, max, sum, or avg. Aggregation can be performed easily in OpenTSDB, PostgreSQL, and KDB+, but BerkelyeDB is difficult because it is a key-value store which does not provide built-in aggregation functions. Therefore, we use Algorithm 2 to benchmark the query for individual time series and compare their performance.

Algorithm 1 Benchmark data loading

```
1: function LOAD( records, insertNumOfRecords)
2:     T ← {}                                          ▷ Initialize the set of throughput
3:     n ← 0                                           ▷ Inserted data point counter
4:     st ← Get the wall clock time
5:     for all r ∈ records do
6:         deviceId, metricId, timestamp, reading, tags ← Get attribute values from r
7:         addRecordToDB(deviceId, metricId, timestamp, reading, tags)
8:         n ← n + 1
9:         if i mod 10000 == 0 then
10:            ed ← Get the wall clock time
11:            t ← 10000/(ed − st)
12:            T ← T ⋃ {t}
13:            st ← ed
14:        if n == insertNumOfRecords then
15:            return
```

Algorithm 2 Benchmark data query

```
1: function QUERY(deviceId, metricId, queryNumOfRecords)
2:     T ← {}                                          ▷ Initialize the set of throughput
3:     n ← 0                                           ▷ Queried record counter
4:     st ← Get the wall clock time
5:     for deviceId ← 0...maxDeviceId do
6:         for metricId ← 0...maxMetricId do
7:             cur ← fetchRecordsOrderByTimestamp(deviceId, metricId)
8:             while hasNext(cur) do
9:                 deviceId, metricId, timestamp, reading, tags ← next(cur)
10:                n ← n + 1
11:                if i mod 10000 == 0 then
12:                    ed ← Get the wall clock time
13:                    t ← 10000/(ed − st)
14:                    T ← T ⋃ {t}
15:                    st ← ed
16:                if n == queryNumOfRecords then
17:                    return
```

3.3 Benchmarking results

We now evaluate the loading and query performance of the four database systems using the air quality IoT datasets. We generate two datasets for our testing: 1) *big data set* contains 1 million sensors (with 10 metrics for each) and 2) *small data set* contains 10 thousand sensors. The reading interval is ten minutes, and the duration of the monitoring is one day (a total of 144 data points). Thus, the total number of data points for the big data set is 1.44 billion (i.e., $1000000 \times 10 \times 144$) and its size is 198.8GB. In the following, we will use the generated data to benchmark the performance of loading, time series queries, and analysis.

Load performance. When loading, we use a bulk with the size of 10,000 for OpenTSDB. For PostgreSQL, we use batch insertion with the same bulk size. For KDB+ and BerkeleyDB, we use their default insert or add method. We measure the performance of loading the big and small datasets, and show the results in Figure 3 and 4, respectively. The loading test uses Algorithm 1, which measures the elapsed time per 10,000 operations and calculates the corresponding throughput. The results are shown as box plots where the red line represents the median of the throughput values, and the height represents the variability. According to the results, KDB+ has the highest throughput, up to 38,000

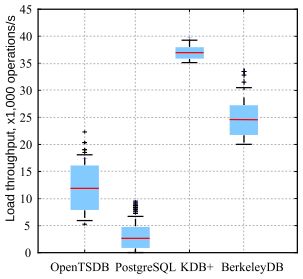

Fig. 3. Throughput of loading big data set

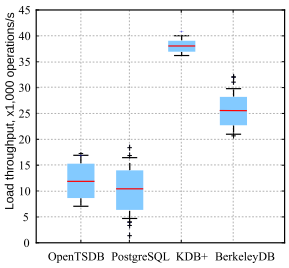

Fig. 4. Throughput of loading small data set

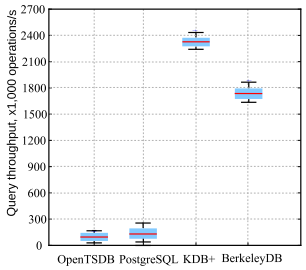

Fig. 5. Throughput of querying big data set

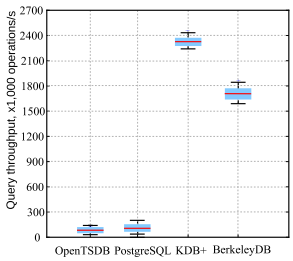

Fig. 6. Throughput of querying small data set

operations per second. PostgreSQL shows a significant difference in loading different sizes of datasets. The throughput of the small data set is 4.3 times the throughput of the big data set. For the investigation, we use the Linux command `vmstat` to check IO and find that for the big data set, the write latency is 5-15% longer than the small data set. In addition, the I/O speed is quite variable, the read speed of 2-15MB/s and the write speed of 0-10MB/s. This may be due to the construction of the index, which involves reading the b-tree pages from disk to memory to update the index. According to the height of box plots, KDB+ has the smallest variability.

We now compare the database sizes after the loading (see Table 1). PostgreSQL is larger than the others because it saves the data in a de-normalized table, and uses extra space for the index. In contrast, the tags in OpenTSDB are saved once in row keys, which saves space. Although it has an additional index table, the table does not seem to cause a significant increase in space usage. KDB+ ranks the second place of using the least space which is due to the data compression in columns. BerkeleyDB hasn't shown space efficiency with its default settings.

Table 1. Database size after the loading the big data set

Raw data	OpenTSDB	PostgreSQL	KDB+	BerkeleyDB
198.8GB	164.8GB	344.5GB	176.4GB	352.8GB

Time series query performance. After the loading experiment, the next step is to benchmark query performance. The query time series is done by using Algorithm 2, which scans each time series with specific metric and device ID. Figure 5 and 6 show the results for the big and small datasets, respectively. The results indicate that the in-memory-based solutions, KDB+ and BerkeleyDB, have better performance for time series query.

Analysis query performance. Analysis query benchmarking is performed only on OpenTSDB, PostgreSQL and KDB+ since Berkeley DB does not provide the operators for analysis queries. We compare three queries on CO_2 emissions, which are shown in Table 2. Different to PostgreSQL and KDB+ which support SQL statements, OpenTSDB queries data through the RESTful interface, and a query is serialized as a JSON object sent over HTTP to the query endpoint (see [16] for more details).

Table 2. Comparison of analysis queries

Analytic Queries	SELECT			WHERE			ORDER BY		
	metric	dev_eui	value	metric	dev_eui	value	metric	dev_eui	value
Total number of readings	ANY			√					
Total CO_2 emission			√	√					
Top CO_2 emission			√	√					√

(i) *Total number of CO_2 readings:* Figure 7 shows the performance of counting the total number of CO_2 readings in the database with the load of the big data set. There are different ways for counting the number of the readings. In this case, we test using "SELECT COUNT(*)" statements without and with specifying the attributes (on metric or value attribute), respectively. The motivation for carrying out this test is to quantify the implication on query performance for different encoding methods used in different columns, e.g., in KDB+. But, for OpenTSDB, it doesn't matter which attribute is chosen for the counting. Therefore, the query times are the same for the three tests, which are around 1.4 seconds. The query time of OpenTSDB is longer than the other two systems. It is interesting to see that the counting on the metric column in KDB+ takes a longer time than the value column. This illustrates the trade-off for achieving better compression for the metric column and the dataset overall, but it takes longer to access the compressed data.

(ii) *Total CO_2 emission:* Figure 8 shows the results for the total CO_2 emission query. The times used by OpenTSDB and KDB+ increase steadily with more sensor data added (indicated as the number of sensors). The performance of KDB+ far outperforms the other two systems, due to its memory-based technology.

(iii) *Top CO_2 emission:* Figure 9 shows the results for the analysis of top CO_2 emission. OpenTSDB and KDB+ show better performance than PostgreSQL, and the times increase slightly with more sensor data. In contrast, the time by PostgreSQL increases faster and almost linearly to the data size. In addition, PostgreSQL uses more time when querying top CO_2 emission, as compared to the time of querying total CO_2 emission in Figure 8, because additional time is required for ordering.

Discussion. Based on the results of the above experiments, we can observe that the memory-based column store, KDB+, indicates the best load and query performance for managing IoT data. KDB+ is a non-distributed database solution suitable for real-time analysis of high-frequency time series data, e.g., IoT and trading data. Key-value stores (e.g., BerkeleyDB), also displays good performance, but lack analysis operators, while these operators are often needed for providing data as a service (DaaS) to other applications. IoT data are generated by many sensors, with the characteristics of fine granularity, high frequency, high volume, and high dimensionality. Comparatively,

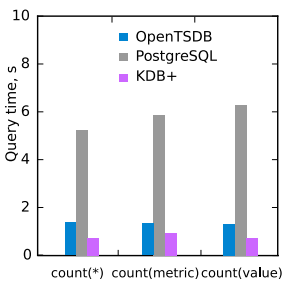

Fig. 7. CO_2 reading number

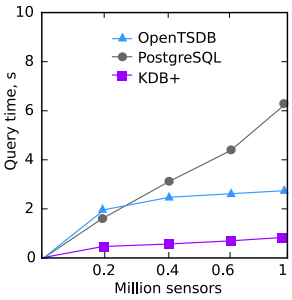

Fig. 8. Total CO_2 emission

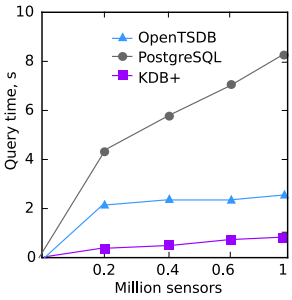

Fig. 9. Top CO_2 emission

it is challenging to use traditional databases, e.g., PostgreSQL, to manage this type of datasets. The distributed open source time series database, OpenTSDB, has shown good performance in both loading and query. Furthermore, OpenTSDB offers built-in analytic operators for manipulating time series, which well suit our needs in the implementation of our monitoring system, i.e., open source, scalable, and good performance.

There are some limitations on this benchmarking work. Among them, OpenTSDB is a distributed database system. Technically, it is unfair to compare with the others on the single-server setting. Second, all database systems are evaluated under their most basic settings, i.e., without advanced optimizations. There is still much room for optimizing the performance for each system. For example, in PostgreSQL, the applicable optimizations can be tuning different configuration settings, table partitioning, clustered indexed, and using a normalized table, etc. For KDB+, it can use a partitioned table; and for OpenTSDB, it can also tune different database settings, and ideally be tested on a cluster. In addition, parallelism is worthwhile for the test, as well as testing more advanced analysis queries (e.g., time series or pattern similarity). However, through this work, we have obtained the initial results of choosing a cloud database for managing IoT data. Obviously, there are a lot of variables that can be considered to obtain comprehensive benchmarking results, for space reason we would like to leave this as our future work.

4 Related Work

IoT-based air quality monitoring has received intensive research effort in recent years, e.g., [4, 7, 17, 9, 22]. These efforts focus on the implementation of the IoT, whilst rarely introduce how to manage IoT data. In contrast, we detail IoT data management, as well as benchmark the alternative technologies. Several solutions have been proposed to manage IoT data. Li et al. present a NoSQL-based storage system IOTMDB to manage scalable IoT data and discuss how to store large-scale data efficiently [12]. However, it is unclear how the data is visualized and accessed. Pintus et al. present a social web-based system [20] of using MongoDB to store IoT data expressed in form of key-value pairs. Ding et al. conversely use the relational data management system PostgreSQL to store sensor data [6]. Di et al. present a document-oriented data model for storing heterogeneous IoT data [5]. Contrary to these works, we take the air quality IoT data management as an example, but emphasize the evaluation of IoT data management technologies. We have conducted extensive experiments to compare different types of

database technologies: SQL and NoSQL, column and relational based, memory and non-memory.

There is a large number of studies addressing database performance. Van et al. evaluate IoT data management on PostgreSQL, Cassandra, and MongoDB [24]. However, the experiments are run between a physical server and a virtual machine, rather than in a real cloud environment. Goldschmidt et al. evaluate the scalability and robustness of three open source time series databases (OpenTSDB, KairosDB and Databus) in a cloud environment, and the results indicate that KairosDB is the best to meet the initial assumptions about scalability and robustness [8]. Sanaboyina et al. evaluate the time series databases OpenTSDB and InfluxDB in terms of energy consumption when database read/write operations are carried out. Phan et al. compare the performance and complexity of NoSQL databases with SQL databases (including MySQL, MongoDB, CouchDB and Redis) [19]. IoT data (sensor readings) and multimedia data are used for their evaluation. The comparison mainly focuses on MongoDB and MySQL, and shows that MongoDB has better scalability. They conclude that scalability is a key point that could potentially make NoSQL better than SQL databases. In our previous work [14, 13], we evaluate the database systems and techniques for managing energy time series data, including traditional, columnar store, and distributed databases. This work focuses primarily on the ability to analyze the well-formatted energy consumption data. IoT data are usually more complicated, for example, with different types, formats, and complex metadata; and the data are often used by IoT applications for different purposes. In addition, there is still a lack of technical work on the evaluation of memory-based technologies for IoT data management. This paper bridges this gap by evaluating the performance, and compares it to traditional and distributed database technologies.

5 Conclusions and Future Work

Air quality monitoring has received increasing attention in recent years. Smart cities necessitate effective tools for air quality monitoring and data management. In this paper, we have presented an air quality monitoring system for smart cities, and established an open source IoT-based platform including software and hardware. We have proposed a scalable IoT data management solution for data collection, analytics and visualization. For the benchmarking, we have evaluated four state-of-the-art database technologies for managing large-scale IoT datasets, and conducted a comprehensive comparison by experimenting on the server. According to the results, the memory-based column store outperforms the others on a dedicated server environment, while the specialized time-series database can also achieve high-throughput writes and reads.

For the future work, we will further improve the system by adding data analytic capabilities, for example, to show the correlation between air quality and other factors such as traffic and weather conditions. We will conduct a more comprehensive benchmarking of database technologies for managing IoT data, and evaluate more advanced analytic queries.

Acknowledgements. This research is supported by the CTT project funded by Local Governments for Sustainability (LoCaL), and the CITIES project funded by Danish Innovation Fund (1035-0027B).

References

1. Ahlers, D., Driscoll P.A., Kraemer F.A., Anthonisen F.V., Krogstie J.: A Measurement-Driven Approach to Understand Urban Greenhouse Gas Emissions in Nordic Cities. NIK, 2016.
2. Apache Zeppelin, http://zeppelin.apache.org/ as of 2017-06-01.
3. Carbon track and trace, http://www.carbontrackandtrace.com as of 2017-06-01.
4. Chen X., Liu X., Xu P.: IoT-Based Air Pollution Monitoring and Forecasting System. In: Proc. of ICCCS, pp. 257–260, 2015.
5. Di F. M., Li N., Raj M., Das S.K.: A storage infrastructure for heterogeneous and multimedia data in the internet of things. In: Proc. of GreenCom, pp. 26–33, 2012.
6. Ding Z., Xu J., Yang Q.: SeaCloudDM: a database cluster framework for managing and querying massive heterogeneous sensor sampling data. The Journal of Supercomputing, 66(3):1260–1284, 2013.
7. Fioccola G.B., Sommese R., Tufano I., Canonico R., Ventre G.: Polluino: An efficient cloud-based management of IoT devices for air quality monitoring. In: Proc. of Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), pp. 1–6, 2016.
8. Goldschmidt T., Jansen A., Koziolek H., Doppelhamer J., Breivold H.P.: Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In: 7th International Conference on Cloud Computing, pp. 602–609, 2014.
9. Jadhav D.A., Patane S.A., Nandarge S.S., Shimage V.V., Vanjari A.A.: Air pollution monitoring system using Zigbee and GPS module. International Journal of Emerging Technology and Advanced Engineering, 3(9), 2013.
10. Kamal-Chaoui L., Robert A.: Competitive cities and climate change. OECD Regional Development Working Papers, 2009(2):1, 2009.
11. KDB+, https://kx.com/ as of 2017-06-01.
12. Li T.L., Liu Y., Tian Y., Shen S., Mao W.: A Storage Solution for Massive IoT Data Based on NoSQL. In: Proc. of IEEE International Conference on Green Computing and Communications, pp. 50–57, 2012.
13. Liu X., Golab L., Golab W., Ilyas I.F., Jin S.: Smart Meter Data Analytics: Systems, Algorithms and Benchmarking. ACM Transaction of Database System (TODS), 42(1), 2016.
14. Liu X., Golab L., Golab W., Ilyas I.F.: Benchmarking Smart Meter Data Analytics. In: Proc. of EDBT, pp. 385–396, 2015.
15. Olson M.A., Bostic K., Seltzer M.I.: Berkeley DB. In: Proc. of USENIX Annual Technical Conference, FREENIX Track, pp. 183–191, 1999.
16. OpenTSDB, http://OpenTSDB.net/ as of 2017-06-01.
17. Pavani M., Rao P.T.: Real time pollution monitoring using Wireless Sensor Networks. In: Proc. of IEMCON, pp. 1–6, 2016.
18. Payload, https://techterms.com/definition/payload
19. Phan T. A.M., Nurminen J.K., Di Francesco M.: Cloud databases for Internet-of-things data. In: Proc. of GreenCom, pp. 117–124, 2014.
20. Pintus A., Carboni D., Piras A.: Paraimpu: a platform for a social web of things. In: Proc. of the 21st International Conference on World Wide Web, pp. 401–404, 2012.
21. PostgreSQL, https://www.postgresql.org/ as of 2017-06-01.
22. Prasad R.V., Baig M.Z., Mishra R.K., Desai U.B., Merchant S.N.: Real time wireless air pollution monitoring system. ICTACT Journal on Communication Technology, 2(2):370–375, 2011.
23. Sanaboyina T.P.: Performance Evaluation of Time series Databases based on Energy Consumption. Master thesis, Blekinge Institute of Technology, 2016.
24. Van der Veen J.S., Van der Waaij B., Meijer R.J.: Sensor data storage performance: SQL or NoSQL, physical or virtual. In: Proc. of Cloud Computing, pp. 431–438, 2012.