Ronda: Real-Time Data Provision, Processing and Publication for Open Data

Fabian Kirstein[1,2(✉)], Dario Bacher[1], Vincent Bohlen[1], and Sonja Schimmler[1,2]

[1] Fraunhofer FOKUS, Berlin, Germany
{fabian.kirstein,dario.bacher,vincent.bohlen,
sonja.schimmler}@fokus.fraunhofer.de
[2] Weizenbaum Institute for the Networked Society, Berlin, Germany

Abstract. The provision and dissemination of Open Data is a flourishing concept, which is highly recognized and established in the government and public administrations domains. Typically, the actual data is served as static file downloads, such as CSV or PDF, and the established software solutions for Open Data are mostly designed to manage this kind of data. However, the rising popularity of the Internet of things and smart devices in the public and private domain leads to an increase of available real-time data, like public transportation schedules, weather forecasts, or power grid data. Such timely and extensive data cannot be used to its full potential when published in a static, file-based fashion. Therefore, we designed and developed Ronda - an open source platform for gathering, processing and publishing real-time Open Data based on industry-proven and established big data and data processing tools. Our solution easily enables Open Data publishers to provide real-time interfaces for heterogeneous data sources, fostering more sophisticated and advanced Open Data use cases. We have evaluated our work through a practical application in a production environment.

Keywords: Open Data · Big data · Real-time

1 Introduction

The ongoing digitization leads to a growing availability of real-time or very frequently updated data sources. This process is supported by the increasing adoption of smart devices and the growth of the Internet of things. Many of these original data sources are strongly connected to public organizations and administrations, such as urban, energy, transportation, or environment data. Therefore, plenty of such data will be and is supposed to be available as Open Data. This requires to facilitate the technical means and processes to enable the publication of real-time Open Data. Popular open source solutions for Open

H. J. Scholl et al. (Eds.): EGOV 2021, LNCS 12850, pp. 165–177, 2021.
https://doi.org/10.1007/978-3-030-84789-0_12

Data, such as CKAN[1], are mainly designed to serve static metadata and file downloads, usually containing aggregated information over a past period of time. However, the timely and raw provision of data is one central requirement in Open Data [1], which cannot be satisfied with the current solutions. Especially the demand in volume and velocity cannot be handled by plain file downloads. From a technical point of view this creates an intersection between aspects of Open Data and big data. Among other things the field of big data provides concepts and tools to handle data, which cannot be managed by traditional methods. [1] Therefore, our general hypothesis is that, *established open source big data technologies and practices constitute a proper foundation for enabling Open Data portals and ecosystems to consume, process and provide (near) real-time data.*

The focus of our work is the architectural design and implementation of a reusable and extendable big data platform which can hold Open Data collected from heterogeneous data sources. The platform is capable of dynamically analyzing, processing and, if needed, anonymizing the incoming data, which can be supplied as streams or batches. To achieve this, big data methods and architectures are analyzed and selected. Our core contributions are:

(1) We designed a comprehensive architecture for retrieving, processing and providing real-time Open Data based on open source software, well-known industry standards, and established big data architecture paradigms.
(2) We developed a fully working prototype to harvest and disseminate real-time data in the context of a production smart city project. Our solution is available as open source and can be easily applied and extended in various Open Data domains.
(3) Our architecture and prototype can serve as a blueprint for similar real-time Open Data projects, demonstrating a path for the next generation Open Data portals, which act as real-time data hubs, instead of simple file servers.

The remaining part of this paper is structured as follows: Sect. 2 reviews related work in the field of Open Data and big data. Section 3 presents the requirements for real-time Open Data, which are transformed into a system design in Sect. 4. Section 5 presents the implementation and evaluates our approach. In Sect. 6 we draw our conclusions and discuss our findings.

2 Related Work

Our work covers the fields of big data, i.e. real-time data streams, and (linked) open (government) data along with related standards and technologies. Charalabidis et al. [1] survey the domain of Open Data and describe differences and similarities to government data, big data and linked data. Government agencies have a long history of publishing their data as open government data and Open Data portals have adopted linked data principles and serve linked open data on

[1] https://ckan.org/.

a large scale [11,12]. This makes linked open government data an established part of data published on the Web.

Governments or companies operating in the public sector already produce real-time data such as traffic, transport or weather data [7]. This data is often not directly accessible, but used in public use cases, e.g. billboards, displays or web applications, or require registration to access it. There exist a variety of different solutions for building Open Data portals. Some of them, such as CKAN are open source, while others, such as OpenDataSoft[2] are closed source [3]. CKAN provides real-time data features through an extension. At the time of writing, this extension was not updated since 2014. OpenDataSoft provides the possibility to push real-time data to previously uploaded datasets or schedule updates for them. These datasets can be exported by users or visualized in graphs, tables or maps. The open government data platform OGoov allows the communication with different data sources via its Real Time Open Data module[3]. Among other sources, it offers an integration to the publish/subscribe endpoint of the FIWARE Orion Context Broker[4] and allows exposing real-time snapshots as well as historical views of that data. Lutchman et al. [15] describe an architecture for a real-time Open Data portal. Data providers register their data streams to the portal's REST endpoints and periodically push data to the server. Data consumers access the data via dedicated REST endpoints and specify the rate with which updates are received. All of the solutions described above rely on the data publisher pushing data to an existing endpoint. To the best of our knowledge, there exists no open source solution, which registers to existing (real-time) data streams and relies on big data technologies.

The WebSocket[5] protocol is standardized by the IETF as RFC 6455 and allows for bidirectional communication between two applications over TCP. The protocol sequence consists of an opening handshake followed by a bidirectional message exchange. WebSockets are primarily targeted at web applications that need two-way communication without opening multiple HTTP sessions. MQTT[6] is a lightweight publish/subscribe message transport protocol designed for communication in constrained environments such as Internet of things or machine to machine communication. The light-weight nature and low packet overhead compared to HTTP makes it especially suited to be used in environments where space or bandwidth limitations apply.

Big data technologies distinguish between stream and batch processing. In batch processing, data is stored in non-volatile memory before it is processed. This has the advantage that a big amount of data can be processed. In contrast, stream processing uses volatile memory before processing and therefore achieves better performance in time aspects. Yet, only a small amount of data

[2] https://www.opendatasoft.com.
[3] https://www.ogoov.com/en/rtod/.
[4] https://fiware-orion.readthedocs.io/en/master/.
[5] https://tools.ietf.org/html/rfc6455.
[6] https://mqtt.org/.

can be grouped for processing [16]. Two of the most popular big data analysis frameworks are Apache Spark[7] and Apache Flink[8]. They both support batch and stream processing. Using batch processing on datasets, Spark achieves better results in terms of processing time [18]. Comparing stream performance, Flink performs better regarding latency [9,10], while Sparks streaming features show better results in terms of throughput [2]. Hadoop Distributed File System (HDFS) and Apache Cassandra are both commonly used in combination with big data analysis frameworks. While Cassandra acts as a distributed database with its querying language (CQL) [4], HDFS is a distributed file system. Cassandra performs better on highly time-based, structured data, whereas HDFS is used for big files and unstructured data [5,8]. Messaging systems provide the means to manage and organize data feeds between applications in a distributed system [13]. Two of the most common messaging systems are Apache Kafka and RabbitMQ. Both use the publish/subscribe pattern to distribute messages and offer scalability using clusters [6]. Kafka focuses on concurrency and fault-tolerance, and can be used as a persistence unit, and to store long term messages without compromising performance. This makes it a good fit for connecting various batch and streaming services. RabbitMQ uses a lightweight, disk-less approach, which makes it a particular fit for embedded systems [6].

3 Requirements for a Real-Time Open Data Platform

In the following, we describe six high-level requirements of our real-time Open Data platform. They are derived from practical functional and non-functional requirements, and form the basis of our work. They are inspired by established Open Data methodologies, and cover the entire process from acquisition, processing and provision of the data.

(1) **Compliance with Open Data standards and best practices:** The publication of data has to follow established Open Data standards and best practices. This includes the application of the DCAT-AP [17] specification for creating metadata and the facility to publish the metadata on existing Open Data portal solutions. Historic aggregations have to be available in an open and machine-readable format, such as CSV or JSON.
(2) **Throughput and scalability:** Ronda has to support a sufficient throughput, which at least fits to typical Open Data access loads. This holds true for data acquisition, processing and provision.
(3) **Support for different data sources:** Open Data is mostly generated from raw data, which is provided by various types of interfaces and in a plethora of data formats. Hence, Ronda needs to support this variety through expandable and custom connectors and transform it into harmonized representations.
(4) **Support for data processing and analysis:** In many cases, the raw data needs to be processed, aggregated and cleaned before publishing. Especially,

[7] https://spark.apache.org/.
[8] https://flink.apache.org/.

if the raw data includes sensitive or personal information, an appropriate preprocessing is indispensable to comply with regulations and privacy policies. Therefore, Ronda needs to support arbitrary and flexible data processing and analysis.

(5) **Real-time access:** The entire data processing has to be performed in real-time, hence, the data has to be available to the user almost immediately (within seconds) after its retrieval from the original source. A suitable protocol for the machine-readable user interface is required. This constraint does not cover the actual timeliness of the source data, which is out of control for our solution.

(6) **Historic and aggregated access:** The collected data should be preserved to allow for a retrieval of historic information. Hence, the real-time data needs to be aggregated over configurable time periods and bundled into downloadable static batches. This makes the data available for non real-time use and creates a transparent log of the data.

4 Designing a Real-Time Open Data Platform

Based on the high-level requirements and the related work in big data, we present the design and technological base of our solution. It is designed to be operated alongside an Open Data portal. Hence, it does not provide the typical functionalities of an Open Data portal, but extends them. The foundation of our design is the so-called lambda architecture, which combines the advantages of batch and stream processing techniques and consists of three layers: a batch layer, a speed layer and a serving layer [14]. This supports the requirement to offer both, real-time and historical data. Incoming data is fed to the batch and speed layer. The batch layer appends all incoming data to an immutable master and computes batch views by analyzing the incoming data. This allows the analysis of historic, not time-critical data and especially improves the analysis for big datasets. The speed layer analyzes only the most recent data and provides the real-time capabilities. In order to realize this architecture the solutions Spark, HDFS and Kafka are selected as core technologies. As indicated in Sect. 2 Spark serves as batch and speed layer, and provides efficient and scalable analysis and manipulation functionalities for big datasets. HDFS is used as the persistence system, since it works well with unstructured and big data and is integrated into Spark's core library. It is provided via Apache Hadoop and acts as serving layer for batch data. Kafka acts as the principal messaging bus to serve data from and to Spark and can easily be connected to it through an official library. In addition, it offers an additional layer of persistence, which improves the fault-tolerance of the system (see Sect. 2) and operates as the serving layer for real-time data.

In order to achieve a clear separation of concerns, simple extensibility, and developer friendliness, the system is divided into five modules. Each module fulfills a distinct task within the data processing chain and in combination with Spark, Kafka and HDFS they depict the lambda architecture. In the following, they are described. Figure 1 illustrates the entire design.

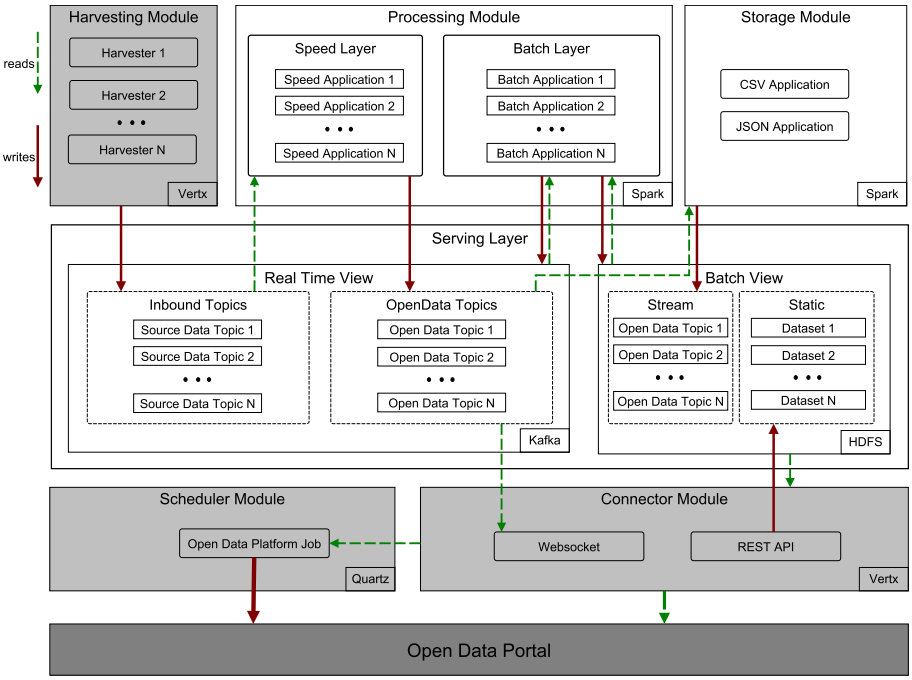

Fig. 1. Architecture overview

The **harvesting module** constitutes the connection to external data sources and main entry point for the data processing. It is responsible for retrieving data from external interfaces, transforming it to an internal representation and injecting it into the Kafka messaging bus. The module offers a flexible extension mechanism to support a broad variety of interface standards. Each data source is mapped to a distinct topic (aka a kind of category) in Kafka.

The **processing module** offers arbitrary and extensive features to alter the source data in real-time. Hence, aggregation, modification, normalization and cleaning tasks are executed here. This also covers any privacy preserving alterations, such as anonymization or data removal. The result of the processing is again stored in a Kafka topic, applying a pre-defined data structure.

The **storage module** is responsible for periodically batching and storing the real-time data. The resulting datasets are chronologically organized and saved in structured data formats, such as CSV and JSON. Frequency, size and resolution of the batches can be individually configured.

The **connector module** represents the public interface of the system, by giving access to the real-time and historic data. Real-time access is provided via the popular and versatile WebSocket protocol. Whereas the static historic data is served via HTTP, supporting content negotiation and query parameters to serve the data in different serialization formats and granularities. The connector

module directly subscribes to the topics in Kafka, which allows for real-time forwarding of the data to the client.

The **scheduler module** functions as a job scheduling system for reoccurring tasks. It is mainly used to connect the system to Open Data portals. Therefore, it can be configured to periodically create metadata (e.g. DCAT-AP-compliant) about available real-time and historic data. In addition, it can be applied for other periodic tasks, such as triggering Spark applications.

5 Evaluation

Based on our design we implemented Ronda as a fully working prototype and evaluated it in two stages: (1) We verified the functional requirements in a real-world production use case, and (2) conducted a comprehensive performance assessment in order to evaluate functionality and scalability.

5.1 Implementation

The practical realization of our design includes the deployment and configuration of the underlying big data tools, and the implementation of the five modules. The harvesting, connector and scheduler modules are custom implementations based on the event-driven and non-blocking Java framework Vert.x[9]. This high-performance framework supports the requirement for scalability and has a strong concept for concurrency and modularization (called Verticles). Moreover, it acts as the main web server to provide the WebSocket and HTTP interfaces. The processing and storage modules are naturally implemented as Spark applications, since they are executed as distributed Spark jobs. The entire system is containerized with Docker and can be easily deployed. In addition, it has support for Kubernetes orchestration[10]. In general, all modules communicate with each other via the Kafka messaging bus. For each data source a separate topic is created, which is used throughout the entire processing chain for a particular source. This horizontal separation for each data source is also represented in each module. For instance the harvesting module is split into distinct submodules for each data source. Furthermore, a common library is available to all modules, providing shared functionalities, such as topic descriptions. The scheduler module makes extensive use of the Quartz scheduling library[11]. The implementation is available as open source[12], which supports a broad application in the Open Data domain.

[9] https://vertx.io/.
[10] https://kubernetes.io/.
[11] http://www.quartz-scheduler.org/.
[12] https://gitlab.com/piveau/piveau-ronda.

5.2 Smart City Use Case

The solution was tested within the research project QuarZ[13], which aims to enable the development of new and innovative smart services for urban districts. The software is being developed for and tested in an urban district in the city of Rüsselsheim am Main that encompasses about 100 homes. As part of the project, households and the surrounding outdoor area were equipped with various sensors and smart devices. The households were provided with smart meters, collecting data about power, gas and water consumption in real-time. Additionally smart home components were offered to all households. In the vicinity, outdoor sensors, collecting temperature, humidity, solar radiation and noise levels were setup. Residents also got access to an electric rental car.

The aim of the project is to collect the generated data into a central data hub. Smart services can then make use of that data while strictly considering the data sovereignty of the individual resident. The central data hub includes an internal storage, as well as an Open Data portal. The internal data hub stores data, emitted by the various sensors and provides the smart services with the desired data. The Open Data portal harvests and integrates traditional Open Data of the region, but in addition publishes real-time data generated in the district for public use. For the latter the internal real-time data streams are retrieved, processed, anonymized, and finally published. We successfully used Ronda for that purpose, by processing data about the power, gas and water consumption of each household, which generates new data points every couple of seconds. In the future, aggregated usage data of the smart home components, the electric rental car and other sources will be integrated too. Based on this data, residents and other interested parties are enabled to create derived services or simply compare their resource consumption with the public.

5.3 Performance Evaluation

We conducted a thorough load test in order to evaluate the performance of Ronda under typical hardware requirements. Therefore, the system was deployed on a Kubernetes cluster. The cluster uses Intel Xeon processors of type E5520 with a base frequency of 2.27 GHz. Table 1 shows the resource consumption for each component.

For HDFS, one name node and four data nodes are deployed as pods on Kubernetes. For Kafka, three Apache Zookeeper instances are required and three Kafka instances are deployed. Zookeeper is a service to provide coordination between the implemented Kafka instances and is mandatory for using Kafka. Spark uses one cluster manager in standalone mode and three worker nodes. The harvesting, processing, storage and scheduler modules are each using one pod. In addition to the Kubernetes cluster deployment, Apache JMeter[14] was installed on a separate client system and used to execute the load tests. Two test

[13] https://www.quartier-der-zukunft.de/ (only available in German).
[14] https://jmeter.apache.org/.

Table 1. System deployment

Component	Instances	RAM	Peek CPU	Volume
HDFS namenode	1	0.8 GB	14 milliCPU	128 GB
HDFS datanode	4	1.2 GB	11 milliCPU	32 GB
Zookeeper	3	0.5 GB	11 milliCPU	–
Kafka	3	3.5 GB	100 milliCPU	16 GB
Spark manager	1	0.5 GB	1 milliCPU	–
Spark worker	3	4.3 GB	22 milliCPU	–
Processing	1	2.4 GB	180 milliCPU	–
Storage	1	1.5 GB	8 milliCPU	–
Connector	5	1 GB	8 milliCPU	–
Harvesting	1	1 GB	6 milliCPU	–
Scheduler	1	0.5 GB	2 milliCPU	–
PostgreSQL	2	0.05 GB	25 milliCPU	5 GB

plans were created within Apache JMeter to measure the performance for both, the user and the data provider view. Suitable test data was programmatically mocked with JavaFaker[15] to create a provider data source. To the best of our knowledge, there does not exist a baseline about expected loads and frequencies on real-time Open Data in the literature. Therefore, we applied the general use statistics of the European Data Portal (EDP) as a rough baseline. The EDP is one of the biggest Open Data portals in the world and in 2018 an average of 1.300 visitors per day was measured [11]. We liberally assume that around 10% of these users will constantly subscribe to a real-time data stream, leading to a threshold of 100 simultaneous requests. Furthermore, we assume a high frequency of incoming data of four values per second.

(1) **Data user view:** The user test plan executed 100 simultaneous HTTP and WebSocket requests for the collected test data. HTTP requests were made on the collected historic test data after two hours of collecting and WebSocket requests were made on the test data source. WebSocket connections remained open as long as no data arrived, afterwards the connections were closed and the time was measured. For HTTP requests, the time was measured after receiving a successful response. The results in Table 2 show an average response time of around 1000 ms and a maximum response time of around 2100 ms, whereas 99% of requests are under 2000 ms. Hence, the system is capable of handling 100 simultaneous WebSocket requests. The HTTP response time was around 1000 ms with 99% under 1929 ms.

(2) **Data provider view:** The test data source sends messages every 250 ms. Four messages per second and 240 messages per minute should be received if the system can process every sent message quickly enough. For this test, the

[15] http://dius.github.io/java-faker/.

connected WebSocket client counted the messages over a period of 10 min. The results are displayed in Table 3. After 600 s all expected 2400 messages are received by the client. This results in a throughput of four messages per second (240 messages per minute).

Table 2. Results of simultaneous requests test

Request	Samples	AVG	95%	99%	Min	Max	Throughput
HTTP	1000	1091 ms	1759 ms	1929 ms	153 ms	2190 ms	48,24159
Websocket	1000	1354 ms	1734 ms	1953 ms	66 ms	2139 ms	49,63518

Ronda is capable of processing the expected 100 simultaneous requests for real-time as well as batch data. Furthermore, the system is capable to support 240 messages from data providers per minute, which is sufficient for typical use cases. However, the evaluation was performed on a specific set of hardware and the underlying architecture scales easily, to ensure the support for much higher amounts of data.

Table 3. Results of processed message test

Request	Estimated time	Messages	Throughput
Websocket	600 s	2400	4

6 Conclusions, Discussion and Future Work

In this paper we have presented Ronda, an open source solution for gathering, processing and publishing real-time Open Data. We have shown that established and industry-proven big data and data processing solutions constitute a proper foundation to enable Open Data publishers to provide and manage real-time interfaces for heterogeneous data sources. Established solutions, such as CKAN, focus on the provision of static file downloads and have little support for real-time data and respective interfaces. Yet, the increased availability of smart devices leads to a rising relevance to provide real-time Open Data, such as weather reports, public transport data, or energy consumption information. Ronda is inspired by the big data lambda architecture and Apache Spark, Kafka and HDFS are used as technological basis. The solution is divided into five highly extendable modules: harvesting, processing, storage, connector, and scheduler. Those modules allow integrating and managing arbitrary real-time sources for Open Data portals and supports the scheduled provision of corresponding metadata. We applied Ronda in a production smart city use case, where residential

power, gas and water consumption data is processed, anonymized and published via a DCAT-AP-compliant Open Data portal. In addition, we conducted a thorough performance evaluation under average hardware requirements. It demonstrated that Ronda can cope with up to 100 simultaneous data consumers for real-time and historic data, while allowing the data providers to push 240 messages per minute. This is sufficient with regard to typical volume, velocity and veracity in the Open Data domain. Therefore, we have shown, that our solution is in compliance with existing Open Data standards, offers sufficient throughput and scalability, supports a variety of data sources, is capable of data processing and analysis, and offers real-time and historic data interfaces.

We are confident, that the Open Data domain will highly benefit from the increasing availability of real-time data sources, since they enable more advanced use cases and business models. Applications and derived products can deliver an advanced experience and functionality over traditional static data. For instance real-time public transportation or traffic applications. Yet, timeliness plays a crucial role, since in many cases the data quickly loses value over time. This clearly shows the relevance to extend the technological foundations of Open Data portals with big data artefacts and establish it as a standard for future implementations. The maturity and capabilities of these industry-proven tools allow a quick transformation of the Open Data domain towards the next level: Open Data portals become powerful data hubs, instead of merely collections of outdated static files. In addition, they provide a variety of features out-of-the-box, such as excellent scalabilty, community support, and a rich extension ecosystem. Beyond that, big data tools constitute a solid basis for even more progressive Open Data applications, such as machine learning or artificial intelligence to further refine the datasets and/or derive additional insights. However, big data tools are complex software artefacts, which add a new level of complexity to Open Data, and currently require more hardware and especially human resources to be implemented and maintained. A joint effort of Open Data developers and publishers is required to establish a common baseline and software recommendations. Ronda can act as a first enabler and starting point for that effort, since it already considers the requirements for real-time Open Data.

In the future, we are planning to extend Ronda with various features and expand our production use case. The actual data streams will be enriched with semantic details, which will increase reusability further. As a showcase, we will provide a data visualization tool, which allows to monitor the real-time data without any detailed technical knowledge. Finally, we aim to further analyze the economic, organizational and social implications of real-time Open Data.

Acknowledgments. This work has been partially funded by the Federal Ministry for Economic Affairs and Energy of Germany (BMWi) under grant no. 01MD18009A ("QuarZ") and by the Federal Ministry of Education and Research of Germany (BMBF) under grant no. 16DII117 ("Deutsches Internet-Institut").

References

1. Charalabidis, Y., Zuiderwijk, A., Alexopoulos, C., Janssen, M., Lampoltshammer, T., Ferro, E.: The open data landscape. In: The World of Open Data. PAIT, vol. 28, pp. 1–9. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90850-2_1
2. Chintapalli, S., et al.: Benchmarking streaming computation engines: Storm, Flink and spark streaming. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1789–1792, May 2016. https://doi.org/10.1109/IPDPSW.2016.138
3. Correa, A.S., Zander, P.O., da Silva, F.S.C.: Investigating open data portals automatically: a methodology and some illustrations. In: Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age, pp. 1–10. dg.o 2018, Association for Computing Machinery, Delft, The Netherlands, May 2018. https://doi.org/10.1145/3209281.3209292
4. Dede, E., Sendir, B., Kuzlu, P., Hartog, J., Govindaraju, M.: an evaluation of cassandra for Hadoop. In: 2013 IEEE Sixth International Conference on Cloud Computing, pp. 494–501, June 2013. https://doi.org/10.1109/CLOUD.2013.31. iSSN 2159-6190
5. Dhruba, B.: HDFS Design (2008). http://svn.apache.org/repos/asf/hadoop/common/tags/release-0.19.2/docs/hdfs_design.pdf. Accessed 01 Mar 2021
6. Dunne, M., Gracioli, G., Fischmeister, S.: A comparison of data streaming frameworks for anomaly detection in embedded systems. In: Proceedings of the 1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec), Orlando, FL, USA (2018)
7. Janssen, M., Matheus, R., Zuiderwijk, A.: Big and open linked data (bold) to create smart cities and citizens: Insights from smart energy and mobility cases. In: International Conference on Electronic Government. pp. 79–90. Springer (2015)
8. Kala Karun, A., Chitharanjan, K.: A review on Hadoop – HDFS infrastructure extensions. In: 2013 IEEE Conference on Information Communication Technologies, pp. 132–137, April 2013. https://doi.org/10.1109/CICT.2013.6558077
9. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE), pp. 1507–1518, April 2018. https://doi.org/10.1109/ICDE.2018.00169. iSSN 1063-6382
10. Kipf, A., Braun, L., Pandey, V., Neumann, T., Böttcher, J., Kemper, A.: Analytics on Fast Data: Main-Memory Database Systems versus Modern Streaming Systems, p. 12 (2017)
11. Kirstein, F., Dittwald, B., Dutkowski, S., Glikman, Y., Schimmler, S., Hauswirth, M.: Linked data in the european data portal: a comprehensive platform for applying DCAT-AP. In: Lindgren, I., et al. (eds.) EGOV 2019. LNCS, vol. 11685, pp. 192–204. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27325-5_15
12. Kirstein, F., Stefanidis, K., Dittwald, B., Dutkowski, S., Urbanek, S., Hauswirth, M.: Piveau: a large-scale open data management platform based on semantic web technologies. In: Harth, A., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 648–664. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_38
13. Korhonen, T.: Using Kafka to Build Scalable and Fault Tolerant Systems, p. 26 (2019)
14. Kumar, Y.: Lambda Architecture – Realtime Data Processing. SSRN Electron. J. (2020). https://doi.org/10.2139/ssrn.3513624

15. Lutchman, S., Hosein, P.: An open source real-time data portal. J. ICT Stand. **2**, 269–302 (2015). https://doi.org/10.13052/jicts2245-800X.235
16. Miloslavskaya, N., Tolstoy, A.: Big data, fast data and data lake concepts. Procedia Comput. Sci. **88**, 300–305 (2016). https://doi.org/10.1016/j.procs.2016.07.439
17. Nuffelen, B.V.: DCAT Application Profile for data portals in Europe (Jun 2020), https://joinup.ec.europa.eu/sites/default/files/distribution/access_url/2020-06/e4823478-4458-4546-9a85-3609867ad089/DCAT_AP_2.0.1.pdf. Accessed 01 Mar 2021
18. Veiga, J., Expósito, R.R., Pardo, X.C., Taboada, G.L., Tourifio, J.: Performance evaluation of big data frameworks for large-scale data analytics. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 424–431, December 2016. https://doi.org/10.1109/BigData.2016.7840633