On Data Persistence Models for Mobile Crowdsensing Applications

Dmitry Namiot[1(✉)] and Manfred Sneps-Sneppe[2]

[1] Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Moscow, Russia
dnamiot@gmail.com
[2] Ventspils International Radioastronomy Centre, Ventspils University College, Ventspils, Latvia
manfreds.sneps@gmail.com

Abstract. In this paper, we discuss various models and solutions for saving data in crowdsensing applications. A mobile crowdsensing is a relatively new sensing paradigm based on the power of the crowd with the sensing capabilities of mobile devices, such as smartphones, wearable devices, cars with mobile equipment, etc. This conception (paradigm) becomes quite popular due to huge penetration of mobile devices equipped with multiple sensors. The conception enables to collect local information from individuals (they could be human persons or things) surrounding environment with the help of sensing features of the mobile devices. In our paper, we provide a review of the data persistence solutions (back-end systems, data stores, etc.) for mobile crowdsensing applications. The main goal of our research is to propose a software architecture for mobile crowdsensing in Smart City services. The deployment for such kind of applications in Russia has got some limitations due to legal restrictions also discussed in our paper.

Keywords: Crowdsensing · Mobile · Cloud · Context-aware

1 Introduction

This article presents an extended and redesigned version of our paper presented on DAMDID/RCDL 2016 conference [1].

As per classical definition, mobile crowdsensing is a sensing paradigm, which is based on the power of the crowd mobile users (more widely - mobile devices) with the sensing capabilities [2]. It is illustrated in Fig. 1.

So, the sensor-enabled devices are the key enablers for mobile crowdsensing. The background for this process is very obvious. There are pure economical reasons behind this process. We see the increasing popularity of smartphones, already equipped with multiple sensors. So, why do not use them for collecting the local timely-based knowledge from the surrounding environment? We do not need to place sensor hardware and we do not need to create special networks for collecting data. That is why, for example, crowdsensing is extremely useful for Smart Cities where budgets are usually limited.

L. Kalinichenko et al. (Eds.): DAMDID/RCDL 2016, CCIS 706, pp. 192–204, 2017.
DOI: 10.1007/978-3-319-57135-5_14

Fig. 1. Mobile crowd sensing [3]

There are several important issues that should be mentioned. Of course, in crowdsensing, we can collect various data: location data, camera information, air pollution data, etc. In other words, it is everything that could be measured with a mobile device. And it is not only that could be done through the sensing features of the mobile device. For example, each smartphone has wireless network modules (Wi-Fi, Bluetooth, Bluetooth Low Energy). We can use information about "visible" (available) wireless networks (nodes, signal strength) as sensing data too. It lets create radio-maps for cities (buildings) and it could be used in physical web models [4]. In other words, the word "sensor" here is the synonym for the word "measurement".

Originally, mobile crowdsensing was about collecting data via smartphones only (as it is presented in Fig. 1). But now we can talk about wearable devices too, about cars for data collecting, etc. For example, our paper [5] presents network proximity models for cars. Classically, GPS data collected by cars have been used to analyze such problems as traffic congestion and urban mobility [6]. As per the latest vision for Internet of Things, we can collect data even from the individual itself – there are so-called cyber-physical systems [7]. Telecom operators, for example, are collecting cellular location information for every SIM-card, so, the mobile phone itself is a sensor collected data about owner's mobility [8].

There are two groups if crowdsensing challenges, usually mentioned in the scientific papers. They are user's anonymity and data sensing quality. The first problem is based on the fact that humans (mobile phone/wearables owners) participate in the process directly or indirectly. In this case, the performance and usefulness of crowdsensing sensor networks depend on the crowd willingness to participate in the data collection process. But we have noted above that the data can be collected not only from the owners of mobile phones. Also, as it was in our case, the information sources may be corporate mobile devices and municipal transport, so that there is no question of voluntariness.

A popular approach for preserving user privacy is depersonalization. It could be done via removing any user identifying attributes from the sensing data before sending it to

the data store. Another approach is to use randomly generated pseudonyms when sending sensed data to the data store [9, 10].

As per [11], we have two categories for mobile sensing: personal and community sensing. Our tasks (Smart City) belong to community sensing. Some of the authors [12] classify participatory and opportunistic sensing. It depends on user participation. Participatory sensing includes the active involvement of users (participants) to contribute data. And opportunistic sensing is more autonomous and should be performed with a minimal user involvement (without it at all).

In our paper we will target another challenge, data stores for mobile crowdsensing. We will present a review of tools (preferably Open Source tools) and architectures used in crowd sensing projects. From this point of view (data stores) there is no difference for participatory and opportunistic sensing.

The rest of our paper is organized as follows. In Sect. 2, we present the common models for crowd sensing data architectures, discuss local databases and cloud-based solution, highlight the importance of stream processing. In Sect. 3, we will discuss crowdsensing for multimedia data (for video applications). In Sect. 4, we discuss backends for Smart Cities. In Sect. 5, we discuss local deployments. Our review has been produced as a part of a research project on Smart Cities and applications in Lomonosov Moscow State University. The main goal of this review is to propose the software architecture for mobile crowdsensing in Smart City environment. We note also that the software architecture for the proposed system must satisfy the existing law restrictions in the Russian Federation, which will be discussed below.

2 Basic Architecture for Mobile Crowdsensing

What are the typical requirements for mobile crowdsensing applications? A good summary can be found in [13]. The basic requirements are as follows:

- provide a minimal intrusion in client devices. The mobile device computing overhead always must be minimized. This requirement should cover all the stages: active state (passing data to data store) and passive state (waiting for new sensing data);
- provide fast feedback and minimal delay in producing stream information. Sometimes it depends on sensors (e.g., for asynchronous sensing [14]), but in general the latency should be minimized;
- provide open interfaces;
- provide security;
- support complete data management workflow, from data collection to communication;

Due to complexity of sensing collecting process, some models propose to use local databases for accumulating data on mobile devices and then replicate them for the processing. This schema is illustrated in Fig. 2 [13].

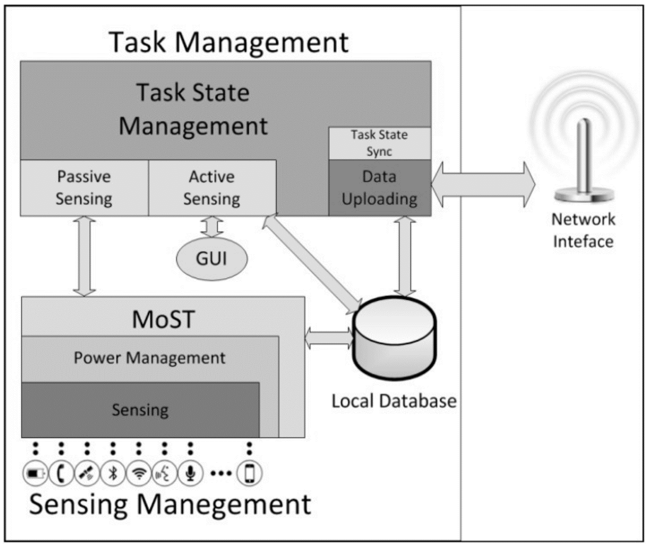

Fig. 2. Local database for sensing

This solution is very easy to implement and there are several options to choose. For example, Android platform offers several options for local data persistence:

- Shared Preferences. This option stores private primitive data in key-value pairs.
- Internal Storage. This option stores private data on the device memory.
- SQLite Databases. It stores structured data in a private database.

SQLite is the most often used solution here. It is a self-contained, embeddable, zero-configuration SQL database engine [15]. For example, Open Source Funf package from MIT [16] saves sensing info in SQLite database (Fig. 3).

Fig. 3. Funf datastore [17]

For crowdsensing system we will get a set of local databases. Accordingly, it will need some process of unification of local data and use a kind of Extract-Transform-Load (ETL) script [18]. Naturally, this model is not suitable for real time processing. As an

intermediate improvement, we can mention saving data from sensors in a cloud-based data store. E.g., the above mentioned Funf package can save data in Dropbox.

The real-time (or near real-time) processing by the scalability reasons in the most cases is associated with some messaging bus. In this connection, we should mention so-called Lambda Architecture [19]. Originally, the Lambda Architecture is an approach to building stream processing applications on top of MapReduce and Storm or similar systems (Fig. 4). Nowadays it is associated with Spark and Spark streaming too [20]. The main idea behind this model is the fact that an immutable sequence of records is captured and fed simultaneously (in parallel) into a batch system and a stream processing system. So, developers should implement business transformation logic twice, once in the batch system and once in the stream processing system. It is possible to combine the results from both systems at query time to produce a complete answer [21].

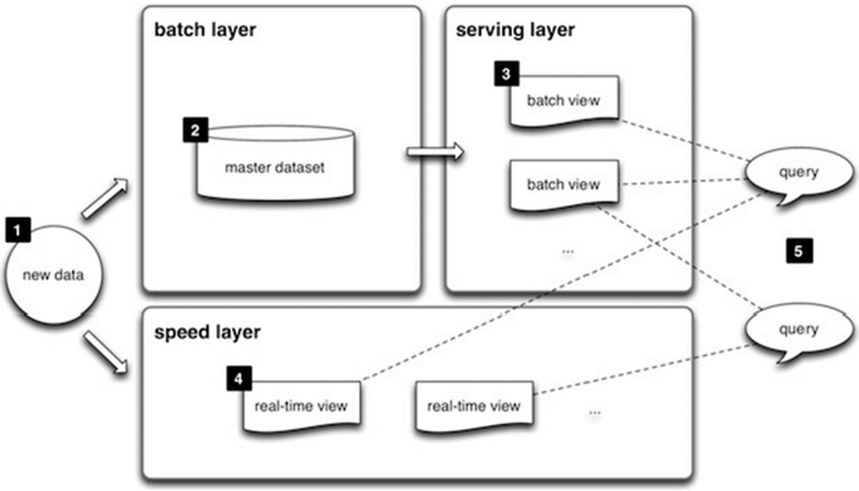

Fig. 4. Lambda architecture [21]

The Lambda Architecture targets applications built around complex asynchronous transformations that need to run with low latency. Any batch processing takes the time. In the meantime, data has been arriving and subsequent processes or services continue to work with old information. The Lambda Architecture offers a dedicated real-time layer. It solves the problem with old data processing by taking its own copy of the data, processing it quickly and storing it in a fast store. This store is more complex since it has to be constantly updated.

One of the obvious disadvantages is the need for duplicating business rules. Practically, the developers need to write the same code twice for real-time and batch layers. One proposed approach to fixing this is to have a language or a framework that abstracts over both the real-time and batch frameworks [22].

The database (data store) design for stream processing has its own specific [23]. In general, there are two options: storing every single event as it comes in (for sensing, it

means storing every single measurement), or storing an aggregated summary of the measurements (events).

The big advantage of storing raw measurements data is the maximum flexibility for the analysis. Hadoop could be used for the full dump. However, the summary (preprocessed data) also has its use cases, especially when we need to make decisions or react to things in real time. Implementing some analytical methods with raw data storage would be incredibly inefficient because we would be continually re-scanning the history of measurements. So, the both options could be useful, they just have different use cases.

As the next step in Lambda architecture, we could mention so-called Kappa architecture [24], where everything is a stream. In our opinion, it is the most suitable approach for sensing data persistence (Fig. 5).

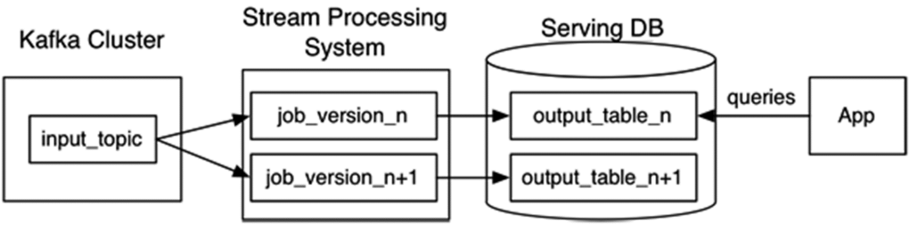

Fig. 5. Kappa architecture

There are several Open Source solutions for data streaming support, e.g., Apache Flink, Flume, Chukwa [19]. However, the most used system (at least, in sensing-related tasks) is Apache Kafka. In general, publish-subscribe architecture is the most suitable approach for scalable crowd sensing applications. Apache Kafka is a distributed publish-subscribe messaging system. It is designed to provide high throughput persistent scalable messaging with parallel data loads into Hadoop. Its features include the use of compression to optimize performance and mirroring to improve availability, scalability. Kafka is optimized for multiple-cluster scenarios [25]. Technically, there are at least three possible message delivery guarantees in publish-subscribe systems:

- At most once. It means that messages may be lost but are never redelivered;
- At least once. It means messages are never lost but may be redelivered;
- Exactly once. It means each message is delivered once and only once.

As per Kafka's semantics when publishing a message, developers have a notion of the message being "committed" to the log. As soon as a published message is committed, it will not be lost. Kafka is distributed system, so it is true as long as one broker that replicates the partition to which this message was written is still alive. In the same time, if a crowdsensing client (producer in terms of publish-subscribe systems) attempts to publish a new measurement and experiences a network error, it cannot be sure when this error happens. Did it happen before or after the message was committed? The most natural reaction for the client is to resubmit the message. It means that we could not guarantee the message had been published exactly once. To bypass this limitation we need some sort of primary keys for inserted data, which is not easy to achieve in

distributed systems. For crowdsensing systems, we can use producer's address (e.g. MAC-address or IMEI of a mobile phone) as a primary key.

Kafka guarantees at-least-once delivery by default. It also allows the user to implement at most once delivery by disabling retries on the producer and committing its offset prior to processing a batch of messages. Exactly-once delivery requires co-operation with the destination storage system (it is some sort of two-phase commit).

In connection with Kafka, we would like to highlight two approaches. The rising popularity of Apache Spark creates the big set of projects for Kafka-Spark integration [26]. And the second approach is the relatively recently introduced Kafka Streams. Kafka models a stream as a log, that is, a never-ending sequence of key/value pairs. Kafka Streams is a library for building streaming applications, specifically applications that transform input Kafka topics into output Kafka topics (or calls to external services, or updates to databases, or whatever). It lets you do this with concise code in a way that is distributed and fault-tolerant [27].

On the client side for crowd sensing applications we could recommend the originally proposed by IBM Quarks System. Now it is Apache Edgent, previously known as Apache Quarks [28]. Quarks System is a programming model and runtime that can be embedded in gateways and devices. It is an open source solution for implementing and deploying edge analytics on varied data streams and devices. It can be used in conjunction with open source data and analytics solutions such as Apache Kafka, Spark, and Storm.

As for the future, there is an interesting approach from a new Industry Specification Group (ISG) within ETSI, which has been set up by Huawei, IBM, Intel, Nokia Networks, NTT DOCOMO and Vodafone. The purpose of the ISG is to create a standardized, open environment which would allow for efficient and seamless integration of applications from vendors, service providers, and third-parties across multi-vendor Mobile-edge Computing platforms [29]. This work aims to unite the telecom and IT-cloud worlds, providing IT and cloud-computing capabilities within the Radio Access Network. Mobile Edge Computing proposes co-locating computing and storage resources at base stations of cellular networks. It is seen as a promising technique to alleviate utilization of the mobile core and to reduce latency for mobile end users [30].

We think also that 5G networks should bring changes to the crowdsensing models. In is still not clear, what is a killer application for 5G. One from the constantly mentioned approaches is so-called ubiquitous things communicating. The hope is that 5G will provide super fast and reliable data transferring approach [31]. Potentially, it could change the sensing too. 5G should be fast enough, for example, to constantly save all sensing information from any mobile device in order to use them in ambient intelligence (AMI) applications [32]. Actually, in this model crowdsensing is no more than a particular use-case for ambient intelligence. But at the moment, these are only theoretical arguments.

3 Crowdsensing for Multimedia Data

In this section, we would like to discuss crowdsensing for video data. The practically considered task was the processing of data security cameras. The key question here is cloud storage for video data. Almost all existing projects related to media data use Amazon Simple Storage Service (S3) (Fig. 6).

Fig. 6. Amazon S3 storage [33]

The storage is based on buckets. Buckets and objects (e.g., videos) are resources managed via Amazon S3 API. The typical applications stores media objects in S3 buckets and keeps keys for them in a separate database (it could be a relational database or NoSQL key-value store). There are many open source applications based on Amazon S3 API, but the key question here is Amazon S3 or its analogs. In Russia, one can face law restrictions with public clouds. As per law, personal data should be stored locally. And since the word "personal" has a very broad interpretation, in practice it means that all locally obtained data should be stored locally too. From existing Amazon S3 analogs we know about partial implementation from Selectel [34]. So, in our opinion, it is a very prospect task for Russia to select some Open Source platform for IaaS and build an own cloud. There are several prototypes for the beginning. As Open Source platforms candidates in this area, we can mention, e.g., CloudStack [35]. Apache CloudStack is an open source cloud computing software, which is used to build Infrastructure as a Service (IaaS) clouds by pooling computing resources. Apache CloudStack manages computing and networking as well as storage resources.

Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) [36] is free and open-source computer software for building Amazon Web Services (AWS)-compatible private and hybrid cloud computing environments.

The OpenStack project [37] is a global collaboration of developers and cloud computing technologists producing the open standard cloud computing platform for both public and private clouds. In our opinion, it is a most suitable choice. OpenStack has a modular architecture with various code names for its components. The most interesting component (in the context of this paper) is OpenStack Compute (Nova). It is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to

manage and automate pools of computer resources and can work with widely available virtualization technologies. It is an analog for Amazon EC2.

OpenStack Object Storage (Swift) is a scalable redundant storage system [38]. With Swift, objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. It lets scale storage clusters scale horizontally simply by adding new servers. Swift is responsible for replication its content.

In our opinion, the cloud-based solution for video data in Smart City applications is a mandatory part of eco-system and OpenStack Swift is the best candidate for the platform development tool.

The importance of cloud-based video services is confirmed by the industry movements. For example, we can mention here IBM's newest (2016) Cloud Video Unit business [39]. As a good example (or even a prototype for the development), we can mention also Smartvue applications [40]. In our opinion, the video processing for data from moving cameras (e.g., surveillance cameras in cars) is a new hot crowdsensing area in Smart Cities.

4 Smart Cities Back-Ends

There are several big Open Source projects directly target back-ends for Smart Cities applications. Firstly, it is FIWARE Cloud [41] (Fig. 7).

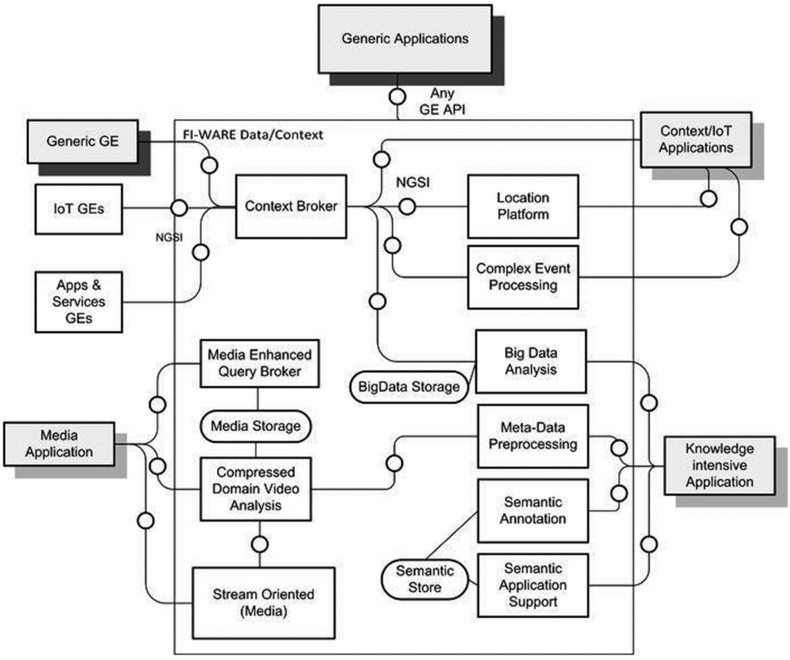

Fig. 7. FIWARE Cloud [41]

Technically, FIWARE supports third-party components (enablers). One of them, for example, proposes generic stream processing. This enabler [42] proposes public API for creating person-to-person services (e.g. video conferencing, etc.), person-to-machine services (e.g. video recording, video on demand, etc.) and machine-to-machine services (e.g. computerized video-surveillance, video-sensors, etc.). However in terms of data storage, it relies on public clouds, like Microsoft Azure.

There is a specific problem with public clouds in Russia: there are almost no developers familiar with the subject. All Smart Cities related projects we know about in Russia always introduce own back-end system. And we do not know even about attempts to create (introduce) a common standard in this area.

In general, it is a perfect example of so-called MBaaS (Mobile Backend As A Service) model for providing the web and mobile app developers with a way to link their applications to backend cloud storage [43]. MBaaS provides application public interfaces (APIs) and custom software development kits (SDKs) for mobile developers. Also, MBaaS provides such features as user management, push notifications, and integration with social networking services. The key moment here is the simplicity for mobile developers. As soon as many (most) of crowdsensing applications rely on mobile phones, this direction is very promising for crowdsensing. Actually, the additional (to data storage) services are the key idea behind MBaaS.

As an Open Source product in this area, we can mention Apache Usergrid [44]. It is composed of an integrated distributed NoSQL database, application layer and client tier with SDKs for developers. Usergrid lets developers rapidly build the web and mobile applications. It provides basic services, such as user registration and management, data storage, file storage, queues, as well as retrieval features such as full text search and geolocation search to power common features for applications.

5 On Practical Use-Cases and Deployment in Russia

In this section, we will present two use cases for back-end systems in prototyping projects with one mobile telecom operator. Firstly, it is wireless proximity information collection. Source data are network fingerprints (list of wireless nodes with signal strength). Each fingerprint has got a time stamp and could be associated (in the most cases) with some geo-coordinates. In our prototype, we use the following chain: Kafka –> Spark Streaming -> Cassandra. Cassandra has been selected as a database suitable for time series. Most of the measurements (including network proximity too) are de-facto time series data (multivariate time series). With the above mentioned chain, we can ensure the compliance with all applicable local restrictions: the personal data will be stored on the territory of the Russian Federation (all the above-mentioned components could be placed in local data centers) and Open Source components provide the absence of claims from the import-substitution point of view (this schema does not use any imported commercial software). Such a bundle is in line with modern approaches, so we can update components, reuse existing open source solutions for them, and participate in developers activities (in Open Source communities around the above-mentioned components).

The second example is much less successful. The idea of the application is data accumulation for dash cameras in vehicles. Data-saving entities are geo-coded media data (so-called ONVIF stream). So, it is crowdsensing for media data. The business value here is absolutely obvious. The existing city cameras are static and they cover predefined areas only, whereas users (cars) in the city can cover all the areas dynamically. De-facto standard for media data in a cloud is Amazon S3. But due to existing regulations (so-called personal data) it could not be used in Russia, because physically data will be saved outside of the country. We think that the "standard" solution is preferable, because there are many available components and systems based on Amazon S3 API. So, even the simulation of this API on the own data model lets reuse many existing software components. In our opinion, with the declared import-substitution and data localization regulations Amazon S3 analogue in Russia should be developed on the national level. Definitely, the only data centers building is not enough and we should talk about software tools for them too. At this moment, this topic is not discussed in Russia, which is a bit strange in our opinion. As a base for S3 analogue development, we can probably use OpenStack (OpenStack Swift). For example, the model from Rackspace Cloud Files (it is an analogue of Amazon S3 and based on OpenStack Swift) [45] could be reproduced.

Currently, Russia starts processes on the standardization of Internet of Things and Smart Cities. Of course, data persistence is an important part of such processes across the world and Russia could not be an exception here. Again, it looks reasonable to reuse already existing developments here. For example, we can mention such projects as oneM2 M or FIWARE (there is a review for domestic standards in our paper [46]). However standards in IoT (M2 M) do not provide dedicated data persistence solutions. They also rely on the existing cloud solutions. So, all the above-mentioned data saving regulations and restrictions are applicable here.

The next important trend is the strategy of vendors of sensors and other measuring devices. Many of them now include data storage as a part of "sensor". For example, Bluetooth tags Eddystone from Google include Google data storage too [47] (unlike iBeacons tags from Apple, for example). In our opinion, this trend will only rise, because data capturing lets vendors provide additional services. It means that with the existing restrictions for data locations, several classes of sensors will hardly be deployed in Russia.

References

1. Namiot, D., Sneps-Sneppe, M.: On crowd sensing back-end. In: DAMDID/RCDL 2016 Selected Papers of the XVIII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2016), CEUR Workshop Proceedings, vol. 1752, pp. 168–175 (2016)
2. Tanas, C., Herrera-Joancomartí, J.: Users as smart sensors: a mobile platform for sensing public transport incidents. In: Nin, J., Villatoro, D. (eds.) CitiSens 2012. LNCS (LNAI), vol. 7685, pp. 81–93. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36074-9_8

3. Internet of Things, Web of Data & Citizen Participation as Enablers of Smart Cities. http://www.slideshare.net/dipina/internet-of-things-web-of-data-citizen-participation-as-enablers-of-smart-cities. Accessed Jan 2017
4. Namiot, D., Sneps-Sneppe, M.: The physical web in smart cities. In: Advances in Wireless and Optical Communications (RTUWO 2015). IEEE Press, New York (2015)
5. Namiot, D., Sneps-Sneppe, M.: CAT - cars as tags. In: 2014 Proceedings of the 7th International Workshop on Communication Technologies for Vehicles (Nets4Cars-Fall). IEEE Press, New York (2014)
6. Massaro, E., et al.: The car as an ambient sensing platform. Proc. IEEE **105**(1), 3–7 (2017)
7. Hu, X., Chu, T., Chan, H., Leung, V.: Vita: a crowdsensing-oriented mobile cyber-physical system. IEEE Trans. Emerg. Top. Comput. **1**(1), 148–165 (2013)
8. Calabrese, F., Ratti, C.: Real time rome. Netw. Commun. Stud. **20**(3-4), 247–258 (2006)
9. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Comput. **1**, 46–55 (2003)
10. Konidala, D.M., Deng, R.H., Li, Y., Lau, H.C., Fienberg, S.E.: Anonymous authentication of visitors for mobile crowd sensing at amusement parks. In: Deng, R.H., Feng, T. (eds.) ISPEC 2013. LNCS, vol. 7863, pp. 174–188. Springer, Heidelberg (2013). doi: 10.1007/978-3-642-38033-4_13
11. Ganti, R.K., Fan, Y., Hui, L.: Mobile crowdsensing: current state and future challenges. IEEE Commun. Mag. **49**(11), 32–39 (2011)
12. Lane, N., et al.: A survey of mobile phone sensing. IEEE Commun. Mag. **48**(9), 140–150 (2010)
13. Bellavista, P., et al.: Scalable and cost-effective assignment of mobile crowdsensing tasks based on profiling trends and prediction: the ParticipAct living lab experience. Sensors **15**(8), 18613–18640 (2015)
14. Namiot, D., Sneps-Sneppe, M.: On software standards for smart cities: API or DPI. In: ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards? Proceedings of the 2014. IEEE Press (2014)
15. Yue, K., et al.: Research of embedded database SQLite application in intelligent remote monitoring system. In: 2010 International Forum on Information Technology and Applications (IFITA), vol. 2. IEEE (2010)
16. Namiot, D., Sneps-Sneppe, M.: On open source mobile sensing. In: Balandin, S., Andreev, S., Koucheryavy, Y. (eds.) NEW2AN 2014. LNCS, vol. 8638, pp. 82–94. Springer, Cham (2014). doi:10.1007/978-3-319-10353-2_8
17. Funf. http://funf.org/. Accessed Jan 2017
18. Bansal, S.K.: Towards a semantic extract-transform-load (ETL) framework for big data integration. In: 2014 IEEE International Congress on Big Data. IEEE Press (2014)
19. Namiot, D.: On big data stream processing. Int. J. Open Inf. Technol. **3**(8), 48–51 (2015)
20. Kroß, J., Brunnert, A., Prehofer, C., Runkler, T.A., Krcmar, H.: Stream processing on demand for lambda architectures. In: Beltrán, M., Knottenbelt, W., Bradley, J. (eds.) EPEW 2015. LNCS, vol. 9272, pp. 243–257. Springer, Cham (2015). doi:10.1007/978-3-319-23267-6_16
21. Lambda architecture. http://lambda-architecture.net/. Accessed Jan 2017
22. Questioning the Lambda Architecture. http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.htm. Accessed Jan 2017
23. Gál, Z., Hunor S., Béla G.: Information flow and complex event processing of the sensor network communication. In: 2015 Proceedings of the 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom). IEEE Press (2015)
24. Merging Batch and Stream Processing in a Post Lambda World. https://www.datanami.com/2016/06/01/merging-batch-streaming-post-lambda-world/. Accessed Jan 2017

25. Garg, N.: Apache Kafka. Packt Publishing Ltd, Birmingham (2013)
26. Maarala, A.I., et al.: Low latency analytics for streaming traffic data with Apache Spark. In: 2015 IEEE International Conference on Big Data (Big Data). IEEE Press (2015)
27. Kafka Streams. http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple. Accessed Jan 2017
28. Apache Edgent. https://edgent.apache.org/. Accessed Jan 2017
29. Mobile-edge computing executing brief. https://portal.etsi.org/portals/0/tbpages/mec/docs/mec%20executive%20brief%20v1%2028-09-14.pdf. Accessed Jan 2017
30. Sanaei, Z., et al.: Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun. Surv. Tutorials **16**(1), 369–392 (2014)
31. Osseiran, Afif, et al.: Scenarios for 5G mobile and wireless communications: the vision of the METIS project. IEEE Commun. Mag. **52**(5), 26–35 (2014)
32. Namiot, D., Sneps-Sneppe, M.: On hyper-local web pages. In: Vishnevsky, V., Kozyrev, D. (eds.) DCCN 2015. CCIS, vol. 601, pp. 11–18. Springer, Cham (2016). doi: 10.1007/978-3-319-30843-2_2
33. Scalable Streaming of Video using Amazon Web Services. http://www.slideshare.net/AmazonWebServices/2013-1021scalablestreamingwebinar. Accessed Jan 2017
34. Selectel API (Russia). https://selectel.ru/services/cloud-storage/. Accessed Jan 2017
35. Apache CloudStack. https://cloudstack.apache.org/. Accessed Jan 2017
36. Nurmi, D., et al.: The eucalyptus open-source cloud-computing system. In: 2009 Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009. IEEE Press (2009)
37. OpenStack. https://www.openstack.org/. Accessed Jan 2017
38. Wen, X., et al.: Comparison of open-source cloud management platforms: OpenStack and OpenNebula. In: 2012 Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). IEEE Press (2012)
39. IBM Cloud Video. https://www.ibm.com/cloud-computing/solutions/video/. Accessed Jan 2017
40. Smartvue. http://smartvue.com/cloud-services.html. Accessed Jan 2017
41. FI-WARE Cloud Hosting. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Cloud_Hosting_Architecture. Accessed Jan 2017
42. Kurento – the stream-oriented generic enabler. https://www.fiware.org/2014/07/04/kurento-the-stream-oriented-generic-enabler/. Accessed Jan 2017
43. Gheith, A., et al.: IBM bluemix mobile cloud services. IBM J. Res. Dev. **60**(2-3), 7:1 (2016)
44. Apache Usergrid. https://usergrid.apache.org/. Accessed Jan 2017
45. Rackspace Cloud Files. https://www.rackspace.com/cloud/files. Accessed Jan 2017
46. Namiot, D., Sneps-Sneppe, M.: On the domestic standards for Smart Cities. Int. J. Open Inf. Technol. **4**(7), 32–37 (2016)
47. Namiot, D., Sneps-Sneppe, M.: On physical web browser. In: Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT), pp. 220–225. IEEE Press, New York (2016)