On Crowd Sensing Back-end

© Dmitry Namiot © Manfred Sneps-Sneppe
Lomonosov Moscow State University, AbavaNet,
Moscow, Russia
dnamiot@gmail.com, manfreds.sneps@gmail.com

Abstract

This paper is devoted to the crowd sensing applications. Crowd sensing (mobile crowd sensing in our case) is a new sensing paradigm based on the power of the crowd with the sensing capabilities of mobile devices, such as smartphones or wearable devices. This power is based on the smartphones, usually equipped with multiple sensors. So, it enables to collect local information from the individual's surrounding environment with the help of sensing features of the mobile devices. In this paper, we provide the review of the back-end systems (data stores, etc.) for mobile crowd sensing systems. The main goal of this review is to propose the software architecture for mobile crowd sensing in Smart City environment. We discuss also the deployment of cloud-back-ends in Russia.

1 Introduction

Crowd Sensing (in our case - Mobile Crowd Sensing) is a relatively new sensing paradigm, which is based on the power of the crowd mobile users (mobile devices) with the sensing capabilities [1]. It is illustrated in Figure 1.

Figure 1 Mobile Crowd Sensing [2]

So, the mobile crowd sensing is all about relying on the crowd to perform sensing tasks through their sensor-enabled devices. The background for this process is very obvious. We see the increasing popularity of smartphones (wearable devices in the nearest future), already equipped with multiple sensors. So, why do not use them for collecting the local timely knowledge from the individual's surrounding environment? In this process, we can collect various data: location data, camera information, air pollution data, etc. In other words, everything that could be done through the mobile device's sensing features. As per the latest vision, we can collect data even from the individual itself – so-called cyber-physical systems [3].

Of course, this approach presents a set of challenges. The main challenges, mentioned in the scientific papers are user participation and anonymity, data sensing quality. Most of the challenges based on the fact that humans participate in the process directly or indirectly. Obviously, the performance and usefulness of crowd sensing sensor networks depend on the crowd willingness to participate in the data collection process. The human participation raises issues regarding the privacy and security of data, as well as issues of revealing of sensitive information [4]. Of course, there are issues regarding the quality and trustworthiness of the contributed data. For example, the big question is how to detect and remove data contributed by malicious users [5]. By the definition, there is no control over the crowd sensors and hence, the system cannot control their behavior. Therefore, the overall quality of the sensor readings may deteriorate if counterfeit data is received from malicious users. Then, the obvious question is how to validate the sensing data that crowd sensors provide to the system. A commonly used approach is to validate the data depending on the trust level of the crowd sensor that reports it [6].

The collection of potentially sensitive information pertaining to individuals is an important aspect of crowd sensing. For instance, sensors readings can be used to track users movements. Such tracks can profile users and this information could be used besides our crowd sensing tasks [7]. A popular approach for preserving users privacy is the 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 [8].

In our paper, we will target another challenge – data stores for mobile crowd sensing. We will present a review of tools (preferably – Open Source tools) and architectures used in crowd sensing projects.

The rest of our paper is organized as follows. In section 2, we present the common models for crowd sensing data architectures. In section 3, we will discuss crowd sensing video applications. In section 4, we

Proceedings of the XVIII International Conference «Data Analytics and Management in Data Intensive Domains» (DAMDID/RCDL'2016), Ershovo, Russia, October 11 - 14, 2016

discuss mobile back-ends. Our review has been produced as part of a research project on Smart Cities and applications for Smart Cities in Lomonosov Moscow State University. The main goal of this review is to propose the software architecture for mobile crowd sensing in Smart City environment. We note also that the architecture of the system must meet the existing restrictions in the Russian Federation, which will be discussed below.

2 The common architecture for mobile crowd sensing

What are the typical requirements for mobile crowd sensing applications? The good summary has been presented in [9], for example. Namely, the requirements are:

- Minimal intrusion on client devices. The mobile device computing overhead always must be minimized. Of course, we should cover all the stages: active state (passing data to data store) and passive state (waiting for new sensing data).
- The fast feedback and minimal delay in producing stream information. It is actually a discussable point of view. Most of the sensors are asynchronous and this fact creates own requirements to gathering data, for example [10]. But in the general – yes, data must be quickly provided.
- Openness and security.
- Complete data management workflow. The application (the platform) should support all steps the data management cycle, from collection to communication.

Due to a complexity of sensing collecting process, some models propose to use local databases for accumulating data on mobile devices and subsequent replication of them. This schema is illustrated in Figure 2 [9].

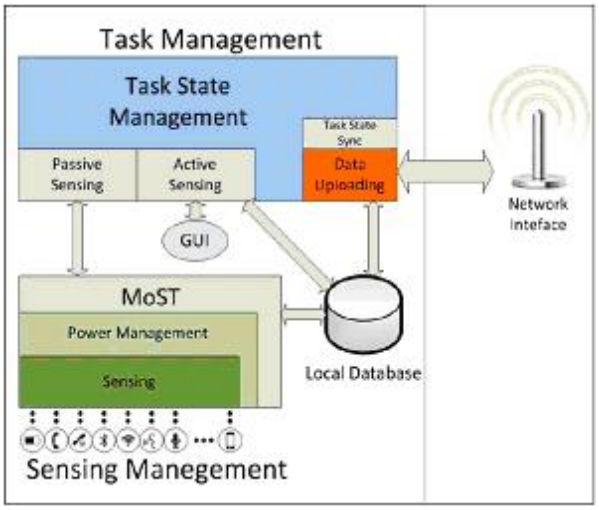

Figure 2 Local database for sensing

For example, Android platform offers several options for local data saving. The solution developers can choose depends on your specific needs, such as whether the data should be private or accessible to other applications (and users) and how much space data requires. Developers can use the following options:

- 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 [11]. For example, Open Source Funf package from MIT [12] saves sensing info in SQLite database (Figure 3).

Figure 3 Funf datastore [13]

So, we can consider crowd sensing system as a set of local databases.

Another popular option in local data stores for sensing is the deployment of cloud-based file stores, like Dropbox [14]. Of course, this architecture does not assume the real-time processing, but it is simple and very easy to implement and deploy.

In the same time, many of the tasks require real-time (or near real-time) processing. In this case, the common use case is associated with some messaging bus. In this connection, we should mention so-called Lambda Architecture [15]. Originally, the Lambda Architecture is an approach to building stream processing applications on top of MapReduce and Storm or similar systems (Figure 4). Nowadays it is associated with Spark and Spark streaming too [16]. The main idea behind this schema is the fact that an immutable sequence of records is captured and fed into a batch system and a stream processing system in parallel. 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 [17].

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 stores it in a fast store. This store is more complex since it has to be constantly updated.

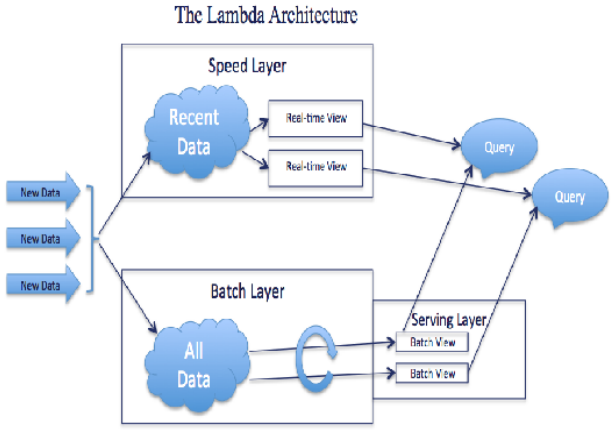

Figure 4 Lambda architecture [18]

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 framework that abstracts over both the real-time and batch framework [19].

The database (data store) design for stream processing has got own specific [20]. Broadly speaking, we have two options:

1. we can simply store every single event as it comes in (for sensing – every single measurement), dump them all in a database or a Hadoop cluster. Now, whenever we need to analyze this data in some way, we can run a query against this dataset. Of course, this will scan over essentially all the events, or at least some large subset of them;
2. we can store an aggregated summary of the measurements (events).

The big advantage of storing raw measurements data is the maximum flexibility for analysis. However, the second option also has its uses, especially when we need to make decisions or react to things in real time. Implementing some analytical methods raw data storage would be incredibly inefficient, because we would be continually re-scanning the history of measurements. The bottom line here is that raw data storage and aggregated summaries of measurements are both could be useful. They just have different use cases.

One of the prospects attempts to combine batch and real-time processing for streams is Apache Flink [21]. Flink has got a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams. It is illustrated in Figure 5.

There are several Open Source solutions for data streaming support. You can find a review in our paper [15]. For example, Flume [22] is a distributed system for collecting log data from many sources, aggregating it, and writing it to HDFS. Chukwa [23] has got similar goals and features.

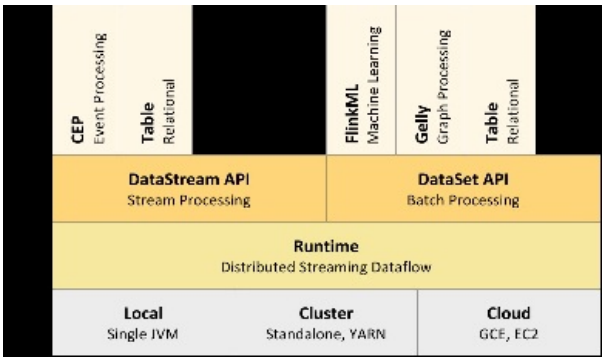

Figure 5 Apache Flink [21]

But the most used system (at least, in sensing tasks) is Apache Kafka. Apache Kafka is a distributed publish-subscribe messaging system. It is designed to provide high throughput persistent scalable messaging. Kafka allows 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 [24]. In general, publish-subscribe architecture is the most suitable approach for scalable crowd sensing applications. Technically, there are at least three possible message delivery guarantees in publish-subscribe systems:

1. At most once. It means that messages may be lost but are never redelivered.
2. At least once. It means messages are never lost but may be redelivered.
3. 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. Once 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 crowd sensing 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. Is it happened 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. It is not easy to achieve in distributed systems. For crowd sensing 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 highlight two approaches. The rising popularity of Apache Spark creates the big set of projects for Kafka-Spark integration [25, 26]. And second, is the 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 recently proposed by IBM Quarks System [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 can be used in conjunction with open source data and analytics solutions such as Apache Kafka, Spark, and Storm (Figure 6).

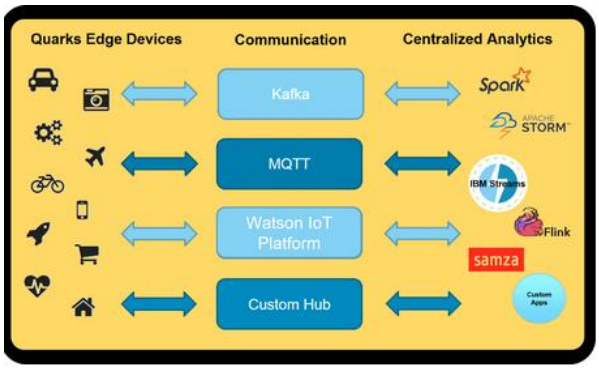

Figure 6 Quarks

As per 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 will allow the 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 crowd sensing models. In is still not clear, what is a killing 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 crowd sensing is no more than a particular use-case for ambient intelligence. But at the moment, these are only theoretical arguments.

3 Crowd sensing for video data

In this section, we would like to discuss crowd sensing for video data. From the practical point of view, the key question here is cloud storage. Almost all existing projects use Amazon Simple Storage Service (S3) for media data (Figure 7)

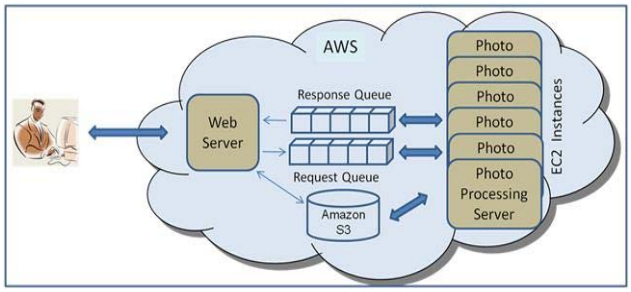

Figure 7 Amazon S3 storage

Amazon S3 is cloud storage for the Internet. It is based on the conception of buckets. To upload your data (photos, videos, documents etc.), you first create a bucket in one of the AWS regions. You can then upload any number of objects to the bucket. In terms of implementation, buckets and objects are resources, and Amazon S3 provides APIs for managing them.

Let us see, for example, the typical mobile crowd sensing application presented in [33].

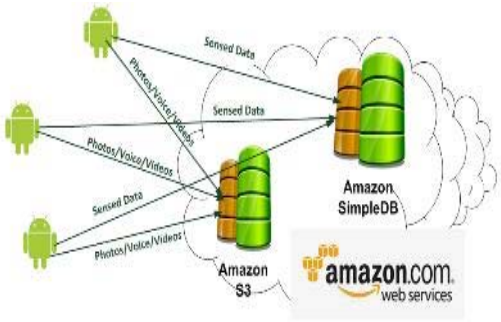

Figure 8 Amazon S3 service on practice

The cloud service provider used for this implementation is Amazon. It uses Amazon SimpleDB, a non-relational highly scalable data store. To store objects namely photos, videos and voice data, it used Amazon S3. The implementation uploads objects to S3 and maintains a key to the upload in SimpleDB. This is a basic solution. Amazon S3 stores media objects and a separate relational database (NoSQL database, e.g., key-value store) keeps keys for objects.

So, the key question here is Amazon S3 or its analogs. With the requirement to store data locally (data should not cross borders) the choice is not big. From existing Russian analogs we know about Selectel [34]. So, the real choice here is to select some Open Source platform for IaaS and build an own cloud. As Open Source platforms in this area, we can mention, for example, 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, 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.

OpenStack has a modular architecture with various code names for its components. We've mentioned just several components which are interested in the context of this paper. OpenStack Compute (Nova) 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.

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

Note, that EU project for Smart Cities platform FIWARE proposes so-called stream generic enabler Kurento [39]. Kurento 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.). But in terms of data storage, it relies on public clouds, like Microsoft Azure.

The importance of cloud-based video services is confirmed by the industry movements. For example, we can mention IBM's newest (2016) Cloud Video Unit business [40]. As a good example (or even a prototype for the development), we can mention also Smartvue applications [41]. By our opinion, the video processing for data from moving cameras (e.g., surveillance cameras in cars) is a new hot crowd-sensing area in Smart Cities.

4 Mobile back-ends

Mobile Backend As A Service (MBaas) is a model for providing the web and mobile app developers with a way to link their applications to backend cloud storage [42]. 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 crowd-sensing applications rely on mobile phones, this direction is very interesting for crowd-sensing. Actually, the additional (to data storage) services are the key idea behind MBaaS.

As an Open Source product in this area, we can mention Convertigo [43]. It lets developers connect to enterprise data using a wide range of connectors such as SQL or Web Services, supports cross-platform development for desktop and mobile apps on multiple devices (iOS, Android), as well as server-side business logic. As another Open Source solution in this area, we can mention FIWARE cloud (Fig. 9)

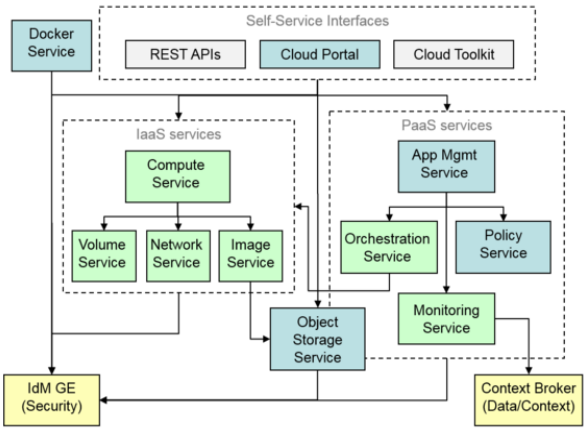

Figure 9 FIWARE mobile cloud [44]

As per [45], MBaaS offerings sit squarely between the existing platform-as-a-service vendors and the full end-to-end solution space occupied by mobile enterprise/consumer application platforms. The basic features for MBaaS include also support for programming device features (e.g., plugins or APIs) such as cameras or sensors, support for development environment (e.g., integrated version control or GIT), visual development tools, multiple operational systems support, cloud deployment, testing support, and activity monitoring. MBaaS should support user authentication (e.g., LDAP, Facebook Connect), mobile applications management, and provide task scheduler for push notifications planning [46].

5 On practical use-cases and deployment in Russia

As a conclusion for this review, we will present two use cases for back-end selection in prototype 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 files (media objects). So, it is crowd-sensing for media data. The business value is transparent – city cameras cover predefined areas only, where 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 out of the country. Alternatively, we can think about Azure Cloud Blob Storage [47] (it is a rival for Amazon S3), but there is the same question about the physical location of data outside of Russia. We think that the "standard" solution is preferable, because there are many available components and systems based on Amazon S3 API [48]. So, even the simulation of this API on the own data model lets reuse many software components. In our opinion, with the declared import-substitution and data localization regulations Amazon S3 analogue in Russia should be developed. Definitely, data centers building is not enough and we should talk about software too. It is a bit strange, why this topic is not discussed. As a base for S3 analogue development, we can probably use OpenStack (OpenStack Swift). For example, as we understand Rackspace Cloud Files is an analogue of Amazon S3 and based on OpenStack Swift.
We would like to highlight also two important moments. The above-mentioned mobile backends are oriented firstly for programming support (e.g., push notifications, social networks support, etc.). They could not solve the problems with data saving regulations, because they are oriented to existing cloud solutions (e.g., Amazon cloud).

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 oneM2M or FIWARE (there is a review for domestic standards in our paper [49]). But standards in IoT (M2M) 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 [50] (dislike iBeacons tags from Apple, for example). In our opinion, this trend will only rise, because data capturing lets vendors to provide additional services. It means that with the existing restrictions for data locations, the whole classes of sensors will be closed to deployment in Russia.

References

[1] Tanas, C., & Herrera-Joancomartí, J. (2013). Users as Smart Sensors: A mobile platform for sensing public transport incidents. In Citizen in Sensor Networks (pp. 81-93). Springer Berlin Heidelberg.

[2] Foremski, P., Gorawski, M., Grochla, K., & Polys, K. (2015). Energy-efficient crowdsensing of human mobility and signal levels in cellular networks. Sensors, 15(9), 22060-22088.

[3] Hu, X., Chu, T., Chan, H., & Leung, V. (2013). Vita: A crowdsensing-oriented mobile cyber-physical system. Emerging Topics in Computing, IEEE Transactions on, 1(1), 148-165.

[4] Ganti, Raghu K., Fan Ye, and Hui Lei. "Mobile crowdsensing: current state and future challenges." IEEE Communications Magazine 49.11 (2011): 32-39.

[5] Hirth, Matthias, Tobias Hoßfeld, and Phuoc Tran-Gia. "Analyzing costs and accuracy of validation mechanisms for crowdsourcing platforms." Mathematical and Computer Modelling 57.11 (2013): 2918-2932.

[6] Chen, Changlong, et al. "A robust malicious user detection scheme in cooperative spectrum sensing." Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012.

[7] Beresford, Alastair R., and Frank Stajano. "Location privacy in pervasive computing." IEEE Pervasive computing 1 (2003): 46-55.

[8] Konidala, Divyan Munirathnam, et al. "Anonymous authentication of visitors for mobile crowd sensing at amusement parks." Information Security Practice and Experience. Springer Berlin Heidelberg, 2013. 174-188.

[9] Bellavista, Paolo, 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 (2015): 18613-18640.

[10] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On software standards for smart cities: API or DPI."

ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?, Proceedings of the 2014. IEEE, 2014.

[11] Yue, Kun, et al. "Research of embedded database SQLite application in intelligent remote monitoring system." Information Technology and Applications (IFITA), 2010 International Forum on. Vol. 2. IEEE, 2010.

[12] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On Open Source Mobile Sensing." Internet of Things, Smart Spaces, and Next Generation Networks and Systems. Springer International Publishing, 2014. 82-94.

[13] Funf Journal http://funf.org/gettingstarted.html Retrieved: Jul, 2016

[14] Novak, Gabor, Darren Carlson, and Stan Jarzabek. "An extensible mobile sensing platform for mhealth and telemedicine applications." Proceeding of Conference on Mobile and Information Technologies in Medicine (MobileMed 2013), At Prague, Czech Republic. 2013.

[15] Namiot, Dmitry. "On Big Data Stream Processing." International Journal of Open Information Technologies 3.8 (2015): 48-51.

[16] Kroß, Johannes, et al. "Stream Processing on Demand for Lambda Architectures." Computer Performance Engineering. Springer International Publishing, 2015. 243-257.

[17] Lambda architecture http://lambda-architecture.net/ Retrieved: Jul, 2016

[18] Simplifying the (complex) Lambda architecture http://voltdb.com/blog/simplifying-complex-lambda-architecture. Retrieved: Jul, 2016

[19] Questioning the Lambda Architecture http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.htm Retrieved: Jul, 2016

[20] Gal, Zoltan, Hunor Sandor, and Bela Genge. "Information flow and complex event processing of the sensor network communication." Cognitive Infocommunications (CogInfoCom), 2015 6th IEEE International Conference on. IEEE, 2015.

[21] Apache Flink http://flink.apache.org/features.html

[22] Waga, Duncan, and Kefa Rabah. "Environmental conditions' big data management and cloud computing analytics for sustainable agriculture." World Journal of Computer Application and Technology 2.3 (2014): 73-81.

[23] Chukwa https://chukwa.apache.org/ Retrieved: Jul, 2016

[24] Garg, Nishant. Apache Kafka. Packt Publishing Ltd, 2013.

[25] Kaveh, Maziar. "ETL and Analysis of IoT data using OpenTSDB, Kafka, and Spark." (2015).

[26] Maarala, Altti Ilari, et al. "Low latency analytics for streaming traffic data with Apache Spark." Big Data (Big Data), 2015 IEEE International Conference on. IEEE, 2015.

[27] Kafka Streams http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple Retrieved: Jul, 2016

[28] Quarks http://quarks-edge.github.io/ Retrieved: Jul, 2016

[29] Mobile-edge computing executing brief https://portal.etsi.org/portals/0/tbpages/mec/docs/mec%20executive%20brief%20v1%2028-09-14.pdf Retrieved: Jul, 2016

[30] Beck, Michael Till, et al. "Mobile edge computing: A taxonomy." Proc. of the Sixth International Conference on Advances in Future Internet. 2014.

[31] Osseiran, Afif, et al. "Scenarios for 5G mobile and wireless communications: the vision of the METIS project." Communications Magazine, IEEE 52.5 (2014): 26-35.

[32] Namiot, D., and M. Sneps-Sneppe. "On Hyper-local Web Pages." Distributed Computer and Communication Networks. Springer International Publishing, 2015. 11-18.

[33] Sherchan, Wanita, et al. "Using on-the-move mining for mobile crowdsensing." Mobile Data Management (MDM), 2012 IEEE 13th International Conference on. IEEE, 2012.

[34] Selectel API (Russia) https://selectel.ru/services/cloud-storage/ Retrieved: May, 2016

[35] Apache CloudStack https://cloudstack.apache.org/ Retrieved: May, 2016

[36] Kumar, Rakesh, and Sakshi Gupta. "Open source infrastructure for cloud computing platform using eucalyptus." Global Journal of Computers & Technology Vol. 1.2 (2014): 44-50.

[37] OpenStack https://www.openstack.org/ Retrieved: May, 2016

[38] Wen, Xiaolong, et al. "Comparison of open-source cloud management platforms: OpenStack and OpenNebula." Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on. IEEE, 2012.

[39] Kurento – the stream-oriented generic enabler https://www.fiware.org/2014/07/04/kurento-the-stream-oriented-generic-enabler/ Retrieved: May, 2016

[40] IBM Cloud Video https://www.ibm.com/cloud-computing/solutions/video/ Retrieved: May, 2016

[41] Smartvue http://smartvue.com/cloud-services.html Retrieved: May, 2016

[42] Gheith, A., et al. "IBM Bluemix Mobile Cloud Services." IBM Journal of Research and Development 60.2-3 (2016): 7-1.

[43] Convertigo http://www.convertigo.com Retrieved: May, 2016

[44] FI-WARE Cloud Hosting https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Cloud_Hosting_Architecture Retrieved: May, 2016

[45] Michael Facemire Mobile Backend-As-A-Service: The New Lightweight Middleware? http://blogs.forrester.com/michael_facemire/12-04-25-mobile_backend_as_a_service_the_new_lightweight_middleware Retrieved: Apr, 2016

[46] Namiot, Dmitry, and Manfred Sneps-Sneppe. "Geofence and network proximity." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. 117-127.

[47] Calder, Brad, et al. "Windows Azure Storage: a highly available cloud storage service with strong consistency." Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011.

[48] Amazon S3 REST API http://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html Retrieved: Jul, 2016

[49] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On the domestic standards for Smart Cities." International Journal of Open Information Technologies 4.7 (2016): 32-37.

[50] Namiot, Dmitry, and Manfred Sneps-Sneppe. "The Physical Web in Smart Cities." Advances in Wireless and Optical Communications (RTUWO), 2015. IEEE, 2015.