Software dependability management in Big Data distributed stream computing systems

Aleksanadar Dimov*, Nikola Davidovic**, Leonid Stoimenov **, Krasimir Baylov*
* Faculty of Mathematics and Informatics, University of Sofia, Bulgaria
** Faculty of Electronic Engineering, University of Niš, Niš, Serbia
aldi@fmi.uni-sofia.bg, {nikola.davidovic, leonid.stoimenov}@elfak.ni.ac.rs, krasimirb@uni-sofia.bg

***Abstract* — Recently there is a rapid increase in application of software intensive systems in the domain of Electric Power Distribution (EPD). EPD companies use a significant number of field devices that are usually connected via a cloud infrastructure. While communicating with each other such devices produce enormous amounts of data that should be processed in order to be of any value for the EPD and the software that is deployed into its infrastructure. Systems that deal with processing of such data, especially in cloud-based environment are denoted as Stream Computing Systems. They usually have very strong quality requirements for reliability, performance, scalability and etc. In this paper we propose an architectural approach to manage dependability and more specifically – reliability of stream computing systems. For this purpose we apply the principles of autonomic computing to an architecture that is based on microservices.**

I. INTRODUCTION

Internet of Things (IoT) is an emerging concept that comprises the interconnection of various hardware devices, such as, end user devices, home appliances, smartphones, field sensors and etc. Such devices are usually connected via a cloud infrastructure that should be based on the concept of microservices [18]. The communication of such devices naturally leads to production of enormous amounts of data, which requires a lot of processing in order to be available for other devices and software systems. Moreover, in many cases this data should be processed simultaneously with real time requirements. The tendency is to have such devices to communicate via software services in cloud based environment. Main purpose of stream computing systems is to make this possible. In this context, usually the quality requirements, both for stream computing systems and the IoT devices are very high and this holds not only for the hardware, but also for the software.

Software quality in terms of dependability becomes an important quality characteristic of contemporary software systems [2]. Dependability has several attributes like availability, reliability, safety, maintainability and etc.

Traditionally, software dependability is determined by the means of software testing [8]. In the context of IoT, however it is very difficult to rely only on testing, because of the natural obstacles that arise for testing into a distributed environment. Additionally, wide class of embedded IoT systems have ultra-high reliability requirements.

Currently there is a trend to increase usage of information systems in the domain of Electric Power Distribution (EPD). One of the main problems of Electric Power Distribution Companies (EPDCs) is to properly and on time have a calculated value of the delivered energy. EPDC in Serbia and its regional center in Nis is obtaining it by having monthly meter readings. Such monthly meter readings are later used to properly bill customers for delivered energy. Lately, with introduction of the smart meters, measures can be remotely read every 15 minutes, so it becomes easier for EPCDs to have meter readouts ready for monthly bills. However, most of the currently used meters are analog devices that just display the currently consumed power. Such meters require human operator to collect the data. But timely data on power consumption are not only needed for the billing. Such data are needed for proper calculation of technical loses and estimation of the commercial loses. Reducing such loses is of the highest importance for the local EPCDs.

In order to obtain accurate and timely data for such analyses, Serbian EPDC's regional center in Niš is introducing various devices that are being used in day to day operations [20]. Each person on the field is equipped with a ruggedized Android based PDA smart device that is used for electric meter readings. Also, the company is purchasing modern grid analyzers, systems that can be used to remotely read various measurements, such as voltage lever, electric currency strength, delivered energy and so on.

Having these devices deployed on the field and set up and connected, it is possible to harvest all needed data to do various kinds of analyses. Based on the meter readings and readings from grid analyzers, it is possible to make an estimation of the level of the technical losses and afterwards make a calculation of the commercial loses. But such readings are usually applied on 15 minutes readings. Making required data to be accurate and received on time is the major requirement of the presented system. Given that such operations are time constrained but rely on the network connection, human input and can reveal sensitive business and personal data of the consumer, the main research question that arises is: how to raise dependability of the results of operation of the future stream computing system that will incorporate analyses of presented 15 minute resolution readings.

The goal of this paper is to define an approach, and a reference architecture for management of reliability of big stream computing system in the EPD domain. The approach presented here is based on the concept of self-adaptation of microservice architecture, as the latter is currently the most common case used in cloud based IoT systems. From purely scientific perspective, it is also of

significant importance, to apply and validate the notion of autonomic computing in different application domains.

The rest of the paper is organized as follows: Section 2 describes some related work in terms of employment of stream computing in EPD; Section 3 presents the theoretical background of our approach; Section 4 shows the main concepts of our architectural solution for software dependability management of EPD stream computing systems and finally Section 5 concludes the paper and gives some directions for further research in the area.

II. Stream Computing in Electric Power Distribution

With the introduction of Smart Grid [1], interdependency of electric power grid and information and communication technology is rapidly emerging [21]. This means that not only electric power grid devices are shifting from analogue technology to digital, which in turns provides more possibility of data measurement and control, it also means that these devices are starting to be interconnected.

But Smart Grid also means that more stakeholders are involved into the power generation and distribution process, hence more data are being gathered and more diversified analyses of such data are needed. Nowadays, dynamic load tracking, dynamic tariffs, clients that can consume but also produce electricity that can be delivered to the grid is becoming a reality [17]. This induces possibilities for massive data collection in different locations of the power grid. But in order to be able to process such high volumes of data, conventional approaches cannot be used anymore. If such data are only stored into the data warehouse and analyzed later, it will not have the same impact as if the analyses are obtained in real-time or in near real time. One of the possible approaches is to use stream computing systems to be able to process such high volumes of data as they come.

Stream computing and stream mining of real-time or near-real-time data is not a novel approach. It is already being used for quite some time in various specific fields such as monitoring of highway traffic [3]. But also there are generic approaches with platforms that can be customized for the real-life deployments, such as S4 [15]. Only lately, after the Smart Grid development has influenced such high data volumes being obtained [7], it is starting to be used in the field of EPD. In [18] authors are identifying the most important technologies that are used in the Smart Grids, namely: Streaming Data Analytics, Load Forecasting, Cloud, Demand Response and Static Analytics. Obviously, first three and the fifth require streaming computing of incoming data to make the system adaptable to dynamical changes that happen throughout the grid. Predicting power supply and demand is one of the big challenges that can be solved using Stream Computing [6]. But not only academia is trying to give answers to such demands for the real-time data analysis, companies like Accenture are already providing solutions that can fulfil such needs [22].

In order to have streaming computing implemented, the underlying infrastructure needs to be able to answer different loads of data that are happening in different time periods. Periods with high consumption but also with high production of electricity can push the hardware to the limits. Such redundancy in the equipment can pose a significant financial requirement. The best answer today is migrating the infrastructure to the cloud. Stream computing big data analytics in Smart Grids is going hand in hand with the Cloud technology [14]. Its characteristics like cost effectiveness, reliability, scalability, availability and elasticity are very useful for such environments. It also means that EPDs can focus on maintaining infrastructure on which their core business is based, namely the power grid and devices that are needed for its functioning. Using the Cloud, the need for the computer hardware infrastructure becomes obsolete.

Most solutions that can be found in the literature are focusing on smart meters that are possible to be read automatically. This is undesired prerequisite since the process of changing meters from the old analogue ones into digital ones is long and tedious and won't be completed in some countries for the foreseeable future. Therefore, alternative approaches should be sought to include manual readings and human input into such big data streaming computing to enable self-adaptiveness and reconfiguration based on the input stream of data from various sources.

III. Theoretical concepts

In this section we are going to briefly present some of the three fundamental notions that are used in our solution – Software reliability and dependability, Autonomic computing and Microservices.

A. *Software reliability and dependability management*

Software reliability is significant software quality parameter, for large variety of systems in different domains. Formally, it is the continuity of correct service delivering, i.e. the belief that a software system will behave as per specification over a given period of time [1]. It is measured by statistical values and may be represented as:

- Probability of failure
- Failure rate
- Mean time to failure

Generally, reliability is part of a broader notion of dependability. The latter is defined as the ability of a computing system to deliver services that can justifiably be trusted [2]. It is depicted by a number of attributes as follows:

- Availability represents readiness of the system to deliver correct service.
- Safety is concerned about absence of catastrophic consequences on the user(s) and the environment in case of system failure.
- Confidentiality is the absence of unauthorized disclosure of information,
- Integrity means absence of improper system state alterations;
- Maintainability is the ability of the system to undergo repairs and modifications.

In the contemporary IoT context reliability is gaining even more attention, because of the high customer and industry requirements towards the distributed smart devices. The most common way to evaluate reliability is to use data from system testing [8]. There exist a number of software reliability models, tailored to process such data and more specifically – number of failures and testing

(or execution) time elapsed between two subsequent system failures [10], [11]. This way practical importance of software reliability is to determine when enough testing is been performed and the system is ready to be shipped to the market. However, it is inherent for all reliability models to make assumptions in order to make estimation possible and such assumptions may reduce applicability of model into development industry.

However, most IoT systems and their constituent parts, like embedded and safety critical systems have ultra-high reliability requirements. Studies have shown that satisfying such requirements is practically infeasible, as one needs to test with uncorrelated test data for thousands of years [4].

This way in many IoT domains, like EPD, alternative approaches for reliability management are needed. In next section we continue with the description of the concept of autonomic computing, which we are going to employ in our reliability management approach.

B. The notion of autonomic computing

Autonomic computing has been introduced several years ago as an attempt to define structural approach towards design and development of dynamic and self-adaptive software systems.

Self-adaptive systems can be viewed from multiple points depending on their direction of autonomy. These points are called self-* properties of the system. They represent the ability of the system to take autonomous actions in the following 4 areas [13]:

- Self-configuration – the system is able to configure itself in order to comply with initially defined high level goals;
- Self-optimization – the system is able to detect and optimize weak points or places that can be improved in terms of specific requirements;
- Self-healing – the system is able to detect problems, create a strategy for fixing them and to apply it;
- Self-protection – the system is able to detect potential threats and defend itself against external attacks.

The idea behind our reference architecture is to design an approach that will target the Self-configuration and Self-optimization properties of the distributed system.

Main aspect of autonomic computing is the so-called control loop (fig. 1).

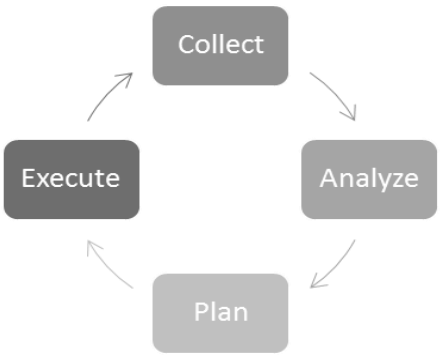

Figure 1. Autonomic Control Loop

The idea behind this is that self-adaptive systems can dynamically change their behavior or properties based on external stimulus. A key mechanism used for running the self-adaptation is the so called control loop or feedback loop [5]. It contains the following main phases:

- Collect – the relevant data is obtained from the surrounding environment and stored for further processing. Usually, this data is collected through agents or sensors.
- Analyze – the data is analyzed and related to existing models.
- Decide – the best action from a list of options is selected
- Act – the system puts the change in place and the loop starts over.

C. Microservices

Service Oriented Architecture (SOA) is well known architectural concept for building distributed software systems that has dominated this software engineering domain during the last two decades. Main reason is that it is claimed to enable reusability and straightforward integration of the main building blocks, called services. However recently, mainly because of quality requirements for system that operate in cloud environment, the architectural style of microservices has emerged [12], [16]. Microservices and SOA share some common characteristics, but also differ in the following:

- Service Granularity – microservices are small and fine-grained. Their small codebase allows developer to quickly identify and fix issues and also easily test them in isolation. In contrast, SOA doesn't provide any recommendation regarding service granularity. Services could vary in size significantly and developers could easily be tempted to build large enterprise services that are hard to maintain.
- Component Sharing – the approach of microservices tend to make each service independent to other shared components. A common technique is to copy the same functionalities to all services and evolve them independently. This approach gives developers freedom to update their services independently by reducing the external dependencies and team coordination. On the contrary, SOA follows the principle of share-as-much-as-possible of given functionality. Although this approach solves the problem with code duplication, it increases the coupling between service components. When a shared component needs to be updated, teams need to analyse the impact on other services and align their release plans to the other services.
- Remote Access Protocols – Microservices favour Representational State Transfer (REST) [9] for exchanging messages. Compared to other protocols REST is relatively simple and lightweight and allows easy ad hoc development and integration. This helps development teams to quickly implement and deploy new services that can exchange messages with others. However, classical SOA doesn't have any specific prescriptions regarding remote access protocols. In result,

protocols are often heavyweight for simple and well-separated applications.

To complement what have been stated above, it should be noted that simplicity is fundamental factor that distinguishes SOA and microservices. The latter strive for simplicity at any level of implementation. Microservices are fine-grained because this way, they are simpler to support. Protocol and message options are reduced, as it is much easier to operate with limited set of interfaces that need to be supported. In consequence, microservice applications are more robust and resilient. They could easily be adapted to new market requirements and attract new customers.

SOA doesn't provide any guideline in terms of simplicity. Although service-oriented systems can be built for simplicity, most times you will encounter large and complex service-based applications. They can integrate with a large set of heterogeneous systems, work with large amount of data and support complex rules for processing messages. But such applications are hard to support and change rapidly. Often such complexity introduces more troubles that solutions.

In general, microservices are a form of SOA. Since SOA practices are significantly less restrictive than microservices it is reasonable to think of microservices as a "form of SOA, perhaps service orientation done right" [12].

In next section we are going to discuss design issues of the system for dependability management based on the notions of microservices and autonomic computing.

IV. Architectural approach for dependability management in EPD smart computing systems

A. System design and analysis

Following the approach of autonomic computing, we should first consider the collection phase. In our proposed system, data is collected using three main sources [19]:

- Smart meters that will be read automatically in 15 minute resolutions.
- Mobile PDAs, that will send data using 3G or WiFi network. The resolution of these readings will rely on the workers on the field.
- Grid analyzer devices that will be deployed on various points of the electric grid. These devices provide data readings in 15 minutes or smaller resolutions.

The presented collection phase can have two potential problems. Namely, reliable and secure data transfer of massive amounts of data form the autonomous working devices such are smart meters and grid analyzers, as well as form the field PDA devices needs to be assured. Human error detection needs to be applied for data received from PDA devices.

Reliable data transfer from smart meters and grid analyzers is addressed by the retailer that needs to provide appropriate technology that assures it. It is in most cases based on the data transfer through the power grid itself. But the reliability problem arises from the grid instability where fault reduction and elimination needs to be applied. Moreover, storing and working with such massive data on the company side, requires proper hardware capabilities, in the form of data warehouses that need to provide mechanisms which prevent data loss. Given that sensitive business data but also data that relate to consumer's privacy is transferred, measures for assuring safe and secure data transfer need to be applied. Since PDA mobile devices depend on the human data insertion, system needs to provide proper signals to the users in cases when inserted values are not within defined threshold. Such threshold can be statically set in advance, but it is better if it is dynamically calculated based on already received data.

When system starts receiving data related to the targeted part of the grid, the analyze phase of the control loop starts. This is made by grid analyzers that are usually deployed on transformer station feeders, but can be also used on any power line that powers some smaller part of the grid. Based on the type of the electricity meters in that part of the grid, metering data are read automatically or are collected by workers carrying mobile PDA devices. Two types of analyses are taking place, first one is primary and addresses the electricity losses by estimating technical and commercial losses. Knowing the power line characteristics and the energy and the electric current on the output as well as the amount of energy delivered to the consumer, it is possible to calculate technical losses. When calculated technical loses and delivered energy are subtracted from the energy measured using the grid analyzer, the amount of commercial loses is determined. The second type of analysis is used for further improvement of the system. All possible values that deviate from the average values for the given period of year are check on the human error. Also the human error warning threshold is calculated and sent back to devices after each block of analysis.

The planning phase relates on the products of the analyses. Based on analysis results, if commercial losses are above the given threshold, the system should plan its reconfiguration in order to narrow down the source of the loses. Whether it can be one customer or one subpart of the grid, in planning phase the system should mark the locations where grid analyzers should be deployed for the next session. Such locations should be shown on the GIS module of the system. In addition, based on the meter types, meter reading locations should be also marked.

By executing the change plan, system will be reconfigured. Given that grid analyzers require physical location changes and proper mounting on the power line as well as the analog meters require presence of workers in the field, we can talk about semi-autonomous execution phase. Once when the devices are properly deployed, the new collection phase can start over. In the meantime, the results in the form of calculations of commercial losses can be consumed in order to prevent them in the future. Generally, losses can come from broken equipment or from theft. Therefore, system execution results should be used by the technical and legal departments in order to mitigate such problems.

B. Micro-services self-adaptive architecture

Next, we will present our microservice-based architecture that implement the aforesaid approach. It considers some of the specifics of microservices like technology heterogeneity, the large number productive services, highly distributed nature, service componentization, etc. Therefore, it can be applied regardless the technology that is used for developing the services. Each of the deployed services could be self-

adaptive. This means that is can monitor itself and the operating environment and change based on internal or external stimulus. The approach is still a work in progress and many of the identified components need to be further scrutinized and described in more details. The proposed architecture is presented in Figure 2 and has a lot of common characteristics with SOA. The reason for this is that microservices and SOA are tightly connected – often considered that microservices is proper way of implementing SOA. However, it is the self-adaptation components that distinguishes our proposed architecture from standard SOA implementations.

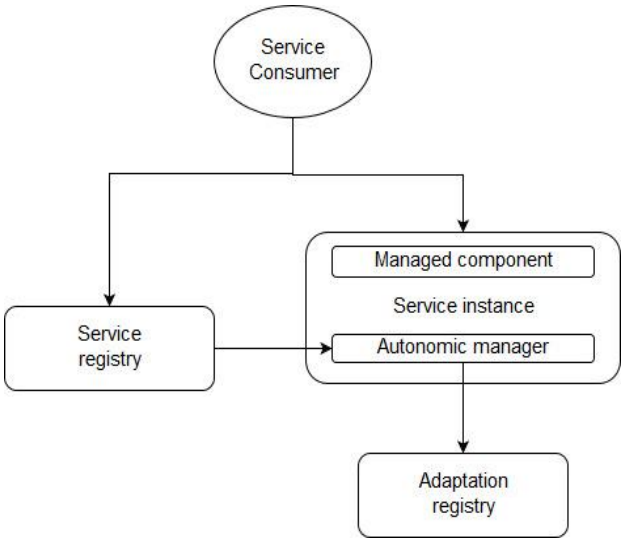

Figure 2. Reference Architecture for Self-Adaptive Microservices in EPD stream computing systems

Below we provide an overview for each of the components shown above.

Service Consumer

Service consumers are the users or systems that consume functionality exposed by the services. For example these are the employees with the PDA devices, other services and/or legacy components of the EPD system or eventually – the smart sensors after they are installed by the EPD Company. Consumers make service invocations in order to use the desired functionality. Service consumers will use REST protocol in order to reach the services.

Service Registry

Service registries serve as a catalog that provides information for the services. This includes information about history of service invocations and about the service itself. History of service invocations would be used to calculate metrics about service quality and specifically to monitor reliability. JDBC would be used to provide metrics data.

Service Instance

The service instance is the actual service implementation and in our architecture consists of two components – managed component and autonomic manager. Managed component contains the functionality of the service instance. Autonomic manager implements the adaptation rules and the data analyzer part of the autonomic control loop. This way it handles service self-adaptation capabilities. It monitors the managed component and apply changes based on the identified adaptation strategy.

Adaptation Registry

The adaptation registry is the key component that allows scaling and quickly expanding the identified adaptation mechanisms. It stores adaptation practices that the service instances can use to manage themselves. Services could search for adaptation practices, via their autonomic managers, based on specific situations or context but they could also register new adaptation strategies based on their knowledge.

V. CONCLUSION

Electric power distribution (EDP) software intensive systems represent a typical Internet-of-Things related domain. EPD companies usually have to collect and process information coming from large amount of distributed smart meters, field sensors and other devices. In this paper we have proposed an architectural approach to manage reliability of software systems that deal with processing of such information. Our solution is based on adaptive architecture, based on microservices, this way making it specifically oriented for execution in cloud environment. Microservice based architecture also fosters scalability of such systems.

Directions for further research include investigations of other quality characteristics like performance. More specifically, efforts should focus on algorithms for caching of information flowing within the system.

ACKNOWLEDGMENT

Research, presented in this paper was partially supported by the DFNI I02-2/2014 (ДФНИ И02-2/2014) project, funded by the National Science Fund, Ministry of Education and Science in Bulgaria.

REFERENCES

[1] Amin, S.M. and Wollenberg, B.F., 2005. Toward a smart grid: power delivery for the 21st century. *IEEE power and energy magazine*, *3*(5), pp.34-41.

[2] Avižienis, A., Laprie, J-C., Randell, B., Basic concepts and Taxonomy of dependable and secure computing, IEEE Trans on Dependable and Secure computing, Vol. 1, Issue 1, Jan -March 2004.

[3] Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H.N., Rahmani, M. and Güç, B., 2010. Real-Time Traffic Information Management using Stream Computing. *IEEE Data Eng. Bull.*, *33*(2), pp.64-68.

[4] Butler, R., and G. Finelli. "The infeasibility of quantifying the reliability of life-critical real-time software." IEEE Transactions on Software Engineering 19.1 (1993): 3-12.

[5] Brun, Y., et al. Engineering Self-Adaptive Systems through Feedback Loops. In Software Engineering for Self-Adaptive Systems. LNCS, Vol. 5525. Springer-Verlag. 2009. 48-70.

[6] Couceiro, M., Ferrando, R., Manzano, D. and Lafuente, L., 2012, May. Stream analytics for utilities. Predicting power supply and demand in a smart grid. In Cognitive Information Processing (CIP), 2012 3rd International Workshop on (pp. 1-6). IEEE.

[7] Diamantoulakis, P.D., Kapinas, V.M. and Karagiannidis, G.K., 2015. Big data analytics for dynamic energy management in smart grids. *Big Data Research*, *2*(3), pp.94-101.

[8] Dimov, A., S. Chandran and S. Punnekkat. (2010). How do we Collect Data for Software Reliability Estimation? Proceedings of the 11th International Conference on Computer Systems and Technologies (CompSysTech). ACM ICPS, vol. 471. Sofia, Bulgaria. June 17-18, 2010. 155-160.

[9] Fielding, R. Architectural styles and the design of network-based software architectures. Diss. University of California, Irvine, 2000.

[10] Farr, W. and M. Lyu, "Software reliability modeling survey" in Handbook of Software Reliability Engineering, New York:McGraw-Hill, pp. 71-117, 1996.

[11] Gokhale, S. "Architecture-Based Software Reliability Analysis: Overview and Limitations", IEEE Transactions on Dependable Security Computing, vol. 4, no. 1, pp. 32-40, 2007.

[12] James, L. and M. Fowler, "Microservices: A definition of this new architectural term", http://martinfowler.com/articles/microservices.html, March, 2014

[13] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." Computer 36.1 (2003): 41-50.

[14] Naveen, P., Ing, W.K., Danquah, M.K., Sidhu, A.S. and Abu-Siada, A., 2016, March. Cloud computing for energy management in smart grid-an application survey. In IOP Conference Series: Materials Science and Engineering (Vol. 121, No. 1, p. 012010). IOP Publishing.

[15] Neumeyer, L., Robbins, B., Nair, A. and Kesari, A., 2010, December. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on* (pp. 170-177). IEEE.

[16] Richards, M., "Microservices vs. service-oriented architecture.", O'Reilly Media, Inc., 2015

[17] Siano, P., 2014. Demand response and smart grids—A survey. *Renewable and Sustainable Energy Reviews*, *30*, pp.461-478.

[18] Sornalakshm, K., Vadivu, G., 2015, A Survey on Realtime Analytics Framework for Smart Grid Energy Management, In *International Journal of Innovative Research in Science, Engineering and Technology*, Vol. 4, Issue 3, pp. 1054-1058, ISSN(Online) : 2319-8753

[19] Stoimenov, L., Davidovic, N., Stanimirovic, A., Bogdanovic, M. and Nikolic, D., 2016. Enterprise integration solution for power supply company based on GeoNis interoperability framework. Data & Knowledge Engineering, 105, pp.23-38.

[20] Stoimenov, L., Davidović, N., Stanimirović, A., Bogdanović, M., Krstić, A. and Nikolić, D., 2011. GinisED Enterprise GIS-Framework for the Utility of the Future. In CIRED 2011, Frankfurt, Germany, 6-9. June.

[21] Taft, J.D. and Becker-Dippmann, A.S., 2015. *The Emerging Interdependence of the Electric Power Grid & Information and Communication Technology* (No. PNNL--24643). Pacific Northwest National Laboratory (PNNL), Richland, WA (United States).

[22] The Right Big Data Technology for Smart Grid – Distributed Stream Computing, https://www.accenture.com/us-en/blogs/blogs-the-right-big-data-technology-for-smart-grid-distributed-stream-computing, last accessed 12.04.2017.