Big Data Assurance Evaluation: An SLA-based Approach

Claudio A. Ardagna, Ernesto Damiani
DI, Università degli Studi di Milano
Crema, 26013, Italy
Email: {firstname.lastname}@unimi.it

Maria Krotsiani, Christos Kloukinas, George Spanoudakis
City, University of London
Northampton Square, London, EC1V 0HB, UK
Email: {Maria.Krotsiani, C.Kloukinas, G.E.Spanoudakis}@city.ac.uk

***Abstract*—The Big Data community has started noticing that there is the need to complete Big Data platforms with assurance techniques proving the correct behavior of Big Data analytics and management. In this paper, we propose a Big Data assurance solution based on Service-Level Agreements (SLAs), focusing on a platform providing Model-based Big Data Analytics-as-a-Service (MBDAaaS).**

***Keywords*-Assurance, Big Data, SLA**

I. Introduction

Big Data is a major research topic, leading all productive environments and enterprises towards the data-driven economy. Better management of data coming from productive processes leads to faster processes, better customer management, and lower overheads/costs. Despite its immense benefits Big Data still suffers from the intrinsic complexity of its technologies and the inability to manage the quality and correctness of the implemented Big Data analytics.

In the last few years, the R&D community focused on providing solutions supporting users in easily implementing a Big Data campaign [1], [2]. Approaches to Big Data Platform-as-a-service have been defined, providing users with a pre-configured Big Data platform, such as in Microsoft Azure and Amazon AWS environments. Users need to only focus on executing the analytics without worrying about how to manage and deploy the Big Data platform. This scenario, however, collides with the lack of expert users having the skills to implement a sound Big Data campaign and retrieve meaningful results. Following this issue, in the last few years, some effort has been done in the definition of techniques supporting the concept of Big Data Analytics-as-a-Service [3], [4], where high-level requirements of the users are transformed in Big Data workflows that can be executed on the target Big Data platform.

In the above context, it is increasingly important to guarantee that the overall Big Data infrastructure complies with users' expectation, and even more with national/international laws and regulations. This challenge points to the concept of *Big Data assurance*, which provides justifiable confidence that a system will behave as expected. In the past, assurance techniques (*i.e.*, audit, certification, compliance) have been used to evaluate the status of distributed systems such as the cloud [5]. These solutions barely apply to Big Data scenarios and require a rethinking of the assurance concept at large.

In this paper, we provide a first approach to Big Data assurance, whereby Service-Level Agreements (SLAs) help evaluate the compliance of a Big Data environment and corresponding analytics, both distributed as a service using a MBDAaaS methodology.

II. Related Work

Assurance techniques have been defined for distributed systems, from service-oriented architectures (*e.g.*, [6], [7]) to cloud environments [8], [9], [10]. These techniques mostly focused on evaluating the security and dependability status of distributed systems. Specifically, They evaluate the status of the system at all layers of the corresponding stack (*e.g.*, service, platform, and infrastructure layers for the cloud), provide a continuous process that evaluates the correctness of the system behavior, and support incremental evaluation to reduce the need of evaluating the system from scratch at each contextual change.

In this paper, we focus on SLAs as a means for supporting Big Data assurance. SLAs have been heavily used to specify terms and conditions for service provision between service consumers and service providers. In order to support the SLA specification process, several specification languages (*e.g.*, [11], [12]) have been defined over the years, with the aim to simplify the SLA specification process for the involved parties and to minimize the time and cost required for it. Despite the extended research on SLA languages and SLA management, SLAs still fail to completely address the requirements of BDA services. Even though new SLA languages have been developed to support cloud services (*e.g.*, SLAC [13], or SLA* [12]), privacy of services is not fully supported. Furthermore, cloud and BDA services support complex operations and have many dependencies between different operations/services, which are difficult to model with current SLA languages.

III. MBDAaaS Methodology

Recently, the research community moved from a paradigm where Big Data application development is driven by the latest technology release, to a more traditional (model-based) paradigm where the users specify their expectations in terms of requirements, and users/consultants follow them

in implementing the Big Data infrastructure. In [3], we presented a Model-based Big Data Analytics-as-a-Service (MBDAaaS) approach, where users specify declarative requirements driving smarter components in carrying out the Big Data processes. MBDAaaS is based on the Model Driven Engineering paradigm and aims to reduce the involvement of the users in Big Data management. The approach in [3] is based on three models as follows.

Declarative model collects a user's requirements and expectations on the Big Data computation. The user defines such requirements in terms of *goals* modeled as a pair (*indicator*, *objective*). An indicator permits to measure the goal, while the objective represents a value the indicator must achieve. Each goal can also be enriched with constraints as pairs (*attribute*, *value*), further refining user's expectations, and prioritized to solve conflicts. Our approach defines declarative requirements along all phases of a Big Data infrastructure instantiation including: *i)* data preparation, *ii)* data representation, *iii)* data analytics, *iv)* data processing, and *v)* data visualization and reporting.

Example. Let us consider a reference scenario provided by Lightsource, a global market leader in energy (solar) networks. Lightsource requires the development of a big data analytics service for calculating the average energy consumption of every appliance for a specific gateway id or household. Data on energy consumption entail hidden sensitive information on the corresponding household. Lightsource can then define the following goals: *i) Anonymization*, with indicator *Anonymization technique* and value *Hashing*, extended with a constraint on the anonymization algorithm such as (*Algorithm*, *SHA-256*); *ii) Analytics Aim*, with indicator *Models* and value *Descriptive* and indicator *Task* and value *Statistical Analysis*.

Procedural model defines a technology-independent workflow, as a service composition, describing how Big Data processes should be carried out to achieve the objectives. It is modeled as a direct acyclic graph where each node represents a service and each arrow models the execution flow.

Example. A procedural model sequentially composes a stream data connector to ingest Lightsource data on energy consumption, a SHA-256 anonymization service, and an analytics service providing descriptive analysis based on an averaging function.

Deployment model specifies how the Big Data processes will work on the target platform. It is a technology-dependent representation of the service composition in the procedural model, using a workflow language (*e.g.*, Oozie) that can be automatically executed on the target platform. The deployment model contains all details on the target system and implemented algorithms, and can be used as a source of assurance requirements.

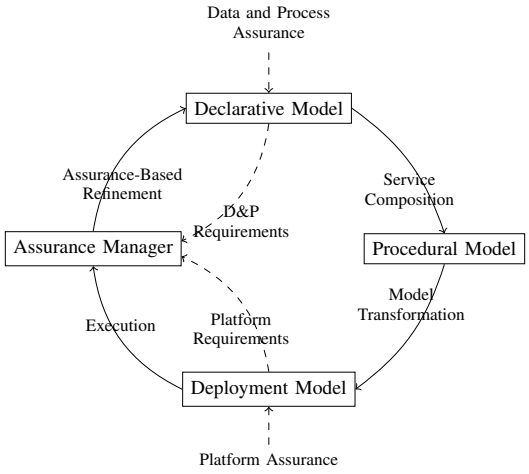

Figure 1. Model-based Big Data Analytics-as-a-Service methodology

Example. The deployment model specifies the details (including the endpoints) of the anonymization and analytics services composed following the procedural model.

The MBDAaaS methodology implements a series of semi-automatic model transformations. The declarative model is used to retrieve a set of services that are compatible with the user goals. For instance, considering goal *Anonymization* above, our methodology retrieves all services implementing a SHA-256 anonymization service. Upon retrieving all services, the user manually configures and composes a subset of them to produce the procedural model. This model is then automatically transformed by a compiler in a deployment model, which is ready to be executed on the target platform. What is still missing in our methodology is the guarantee that the retrieved results are correct, meaning that a sound process addressing the declarative requirements of the user is executed on the Big Data infrastructure. To this aim, Figure 1 sketches how our methodology can be integrated with an assurance process (Section IV).

IV. Big Data Assurance

Big Data assurance extends traditional assurance to evaluate the status of the platform and its behavior, as well as the status of the analytics process and of the corresponding data. The latter points to *veracity* of the 5V model for Big Data [14], which refers to the trustworthiness of data. In this paper, we focus on the assurance evaluation of our MBDAaaS methodology, as follows.

- *Data assurance.* It evaluates how data are managed, represented, and prepared for the Big Data infrastructure. It relates to the two areas of the declarative model regarding data preparation and data representation. For example, sensitive data on energy consumption of households can be anonymized by removing details on the data owner identity by means of hashing techniques

(*e.g.*, SHA-256). An assurance manager should verify whether data are always kept anonymized.

- *Process assurance.* It evaluates the status of a Big Data process, by monitoring all activities between the data ingestion and the production of the analytics outcome. It relates to the two areas of declarative model regarding data analytics and data visualization and reporting. For example, data of EU household might be required to be always stored and processed by components deployed in an EU country. An assurance manager should verify the distribution of data among components to ensure that data never leave EU countries.
- *Platform assurance.* It evaluates the status of the Big Data platform, where Big Data analytics are executed. It also considers the behavior of non-functional components supporting data security and privacy, and process monitoring. For instance, it includes requirements on performance, parallelization, processing, and elasticity. For example, cryptographic techniques should be employed to guarantee data confidentiality (*e.g.*, encryption). An assurance manager should verify that the encryption components behave correctly.

In the following, we define a Big Data assurance process for MBDAaaS, based on SLAs where the declarative model is the main source of requirements and the target of assurance specification and verification.

V. SLA-based Big Data Assurance

SLAs are formal agreements that clarify the service provisioning and the responsibilities between the parties involved, and facilitate the communication between them [15]. At a minimal level, SLAs should define the properties that need to be preserved during the provision of a service and the penalties that will be applied if those properties are violated. In this paper, we extend the WS-Agreement specification language to specify SLAs covering the MBDAaaS concept and securely use Big-Data-Analytics services.

A. SLA Specification

WS-Agreement [11] was introduced by the Open Grid Forum to address key requirements for the specification of SLAs, such as supporting modularity, accommodating other external and domain specific standards, and allowing extensions. In WS-Agreement, an SLA is composed of three main sections: *i)* SLA name (optional), *ii) Context*, containing the meta-data for the entire SLA (*e.g.*, the SLA participants, its lifetime), *iii)* SLA terms that contain *Service Terms* (STs) describing the services regulated by the SLA and *Guarantee Terms* (GTs) specifying the service levels that should be satisfied during the provision of the service.

The limitations of WS-Agreement for our purposes are related to the lack of specification support for: *i)* security and privacy Service Level Objectives (*SLOs*); *ii)* actions that need to be taken during the life cycle of an SLA (*e.g.*, platform modification actions or SLA re-negotiation actions); *iii)* multi-party SLAs; and *iv)* comprehensive models of complex services, composed of internal operations, service assets, data and other dependencies. Thus, to support monitorable SLOs, we extended the sub-element *CustomServiceLevel* of the *ServiceLevelObjective*, by introducing a new type called *PreciseSLOType*, to support the specification of SLOs at:

1) *Declarative level*, based on the declarative model of our MBDAaaS approach (section III). This is specified as a property of a particular category that is applied to a service asset (*e.g.*, internal or external operation or data elements of the service).
2) *Procedural-Deployment level*, which provides the exact run-time assertion for the SLO satisfaction.

As an example, we can extract the information regarding the *Anonymization* security property that should be applied on the sensitive data of households, based on a specified algorithm (*e.g.*, SHA-256). Also, based on the corresponding procedural and deployment models, we can retrieve the information of the service target.

To support the specification of actions that should be undertaken when terms are violated, we extend the WS-Agreement's sub-element *CustomBusinessValue*. Our extension permits to express different types of actions by each GT violation. Two actions have been predefined: *i) renegotiate*, which causes the SLA to be renegotiated, and *ii) penalty*, which causes a penalty (or reward if negative) to be incurred. SLA modellers are also free to use any other action name they wish and can also guard all actions with a predicate. A *guard* predicate declares conditions that should be satisfied, in addition to the GT violations.

B. MBDA-SLA Negotiation Process

SLA negotiation is characterized by the observation that a customer and a service provider are separated by a knowledge gap when initiating the SLA negotiation process. By decreasing this knowledge gap, customers and service providers can successfully negotiate a reasonable SLA for a specific service provision.

Our approach is based on the PROSDIN negotiation framework, a proactive runtime SLA negotiation tool [16]. Our negotiation rules are Jess rules ([17]) defined in XML and are condition-action rules of the form:

$$\mathbf{IF}(condition)\mathbf{THEN}(action)\mathbf{ELSE}(action)$$

Conditions are either atomic conditions or logical combinations of atomic conditions over property attributes of services. Rule actions can be either *accept* or *reject* actions that are used to accept/reject the value of property attributes in a given SLA offer; or *set* actions that are used to propose a new value or range of values for one or more property attributes in a given SLA offer.

To proceed with the SLA Negotiation process, we take the viewpoint of a middleman (*i.e.*, the MBDA platform) between service consumers and service providers. There are two main types of SLAs we consider: pay–as–you–go (PAYG) and subscription types. They both assume an average rate of service use. In PAYG, the consumer pays a price each time a service is used, while in subscription they pay a subscription each interval (week, month, *etc.*). Herein we consider PAYG SLAs only, to simplify the presentation.

We assume that for each consumer c_i we know the:

- C_i^{in}: income received per request;
- P_i^{in}: penalty paid by us when a request is not serviced;
- ${}^{C}R_i^{in}$: contractually agreed/negotiated average rate of service requests; and
- ${}^{O}R_i^{in}$: its observed/assumed average request rate.

Thus, for a new consumer, we might negotiate for a rate ${}^{C}R_i^{in}$, when we assume that they will have a lesser rate ${}^{O}R_i^{in}$. Similarly, for an existing consumer, we would have agreed upon a ${}^{C}R_i^{in}$, but observe that they actually have a lesser rate ${}^{O}R_i^{in}$. It is always the case that ${}^{O}R_i^{in} <= {}^{C}R_i^{in}$, otherwise our SLA would be violated by the consumer. Furthermore, we assume that for each service provider p_j we know the:

- C_j^{out}: income received per request;
- P_j^{out}: penalty paid when a request is not serviced;
- ${}^{C}R_j^{out}$: contractually agreed/negotiated average rate of service requests; and
- ${}^{O}R_j^{out}$: its observed/assumed average request rate.

Our observed/assumed rate ${}^{O}R_j^{out}$ may be greater than ${}^{C}R_j^{out}$, as there is nothing forbidding the provider to serve requests faster than promised. If it is lesser than ${}^{C}R_j^{out}$, then the provider may end up paying a penalty, assuming that consumer requests are forwarded at the agreed rate ${}^{C}R_j^{out}$. However, it is not possible to know the actual provider's rate of service processing or how likely it fails to service a request. But, we can consider the number of lost requests at the provider's side L_j^{out}. It should be stressed that all rates are average rates, so there are situations where they arrive too fast/slow or they can take more/less time to be served than the average. Thus, in order for an SLA to be agreed, we need to know the: *i)* max/min consumer request rate; *ii)* penalty amount; and *iii)* payment for the service.

Example. Based on the *Platform assurance* layer (Section IV), information regarding the deployed components and their configuration can be used regarding the max/min request rate. Reward and penalty amounts should be defined by the SLA modeler. Thus, in order to start a negotiating process we need: *i)* boundaries of the SLA terms that can be agreed; *ii)* penalties and charges to be negotiated. For this reason, we use the Prism tool [18] to retrieve these figures, and facilitate the SLA negotiation process. Since we need the request rate, we will base our formal model on the (finite-buffer) M/M/s/K queue [19] that has an incoming rate of requests λ, s servers with a serving rate μ and a *buffer* with capacity K to help with request/service spikes (since rates are average ones). The PRISM model [18] is a continuous-time Markov chain (CTMC). Listing 1 shows the basic model for consumers and Listing 2 for providers. Each of these has a single transition, which fires with a specific rate. All consumers use the same module definition as consumer 1 (Listing 1) renaming its action and rate – similarly for providers (Listing 2, lines 7–10).

Listing 1. The consumer module

```
1  const double reqRate;
2  module Consumer1
3   [req1] (true) -> reqRate: true;
4  endmodule
```

Listing 2. The provider (server) module

```
1  #const s#
2  const double srvRate;
3  const int s=#s#;
4  module Provider1
5   [srv1] (true) -> srvRate: true;
6  endmodule
7  #for i=2:s#
8  module Provider#i#=Provider1 [srv1=srv#i#]
9  endmodule
10 #end#
```

Listing 3. The MBDAaaS platform module

```
1  const int Capacity;
2  module Queue_M_M_s_K
3    waiting : [0..Capacity] init 0;
4    // Clients drive these actions
5    [req1] (waiting<Capacity) -> 1 : (waiting'=waiting+1);
6    [req1] (waiting=Capacity) -> 1 : true; // lost req
7    // Service providers drive these actions
8  #for i=1:s#
9    [srv#i#] (waiting>=#i#)
10     -> 1 : (waiting'=waiting-1);//serve if enough requests
11 #end#
12 endmodule
```

The module in Listing 3, representing the MBDAaaS platform, links consumers and providers together. The transitions of this module are synchronized with the same-named transitions of the other modules. It defines their rate as 1.0, thus their global rate is the one declared in the other modules (rates of synchronized transitions in PRISM are multiplied together). Each time a consumer makes a request and there are free slots in the waiting queue, the MBDAaaS module accepts the request, incrementing the number of waiting requests. If the waiting queue is full this module rejects the request. It should be noted that the model is parametric on the number s of servers (see lines 8–11), which can be extracted from the deployment model.

VI. Discussion and Conclusions

In this paper, we provided a solution to *Big Data Assurance* based on SLAs, which integrates with a MBDAaaS solution. We applied our solution in a real-world scenario of Lightsource, focusing on the domain of data security and privacy management. Data of energy consumption carry sensitive information that can be used to infer people habits and behavior (*e.g.*, working time, bed time), and pose strict requirements, especially with the advent of the General Data Privacy Regulation (GDPR) in Europe. Privacy and security requirements are specified in the declarative model,

possibly including constraints on the target platform and its configuration. Two main requirements of our evaluation are:

1) *Data anonymization* (data and platform assurance). GDPR (articles 25 and 51.f) mandate users' sensitive data to be protected at a reasonable level of strength against inference attacks and data leaks. A possible approach for data protection is anonymization that can be specified in the data preparation area of the declarative model.
2) *Data confidentiality* and *integrity at rest* (data and platform assurance). GDPR (articles 24 and 25) mandate to safely store users' data, guaranteeing their confidentiality and integrity. Cryptographic techniques could be deployed to guarantee data confidentiality and data integrity.

Driven by the above declarative requirements, an executable deployment model of the Big Data process is generated specifying the details of the integrated privacy services. These high-level requirements are fed into the SLA specification and negotiation tool, to generate the relevant SLAs.

Based on our scenario, a service consumer (*i.e.*, Lightsource) using our MBDAaaS platform generates an SLA and triggers the SLA negotiation process. By providing all requirements needed to generate the three models of our methodology, an SLA offer is produced (section V-A), which is then translated into a PRISM model, to be evaluated by the PRISM tool (section V-B). When the PRISM tool finishes validating the model, the results are translated into Jess rules and are sent to the Jess rule engine, which checks those values and tries to match them with existing values from a service provider(s). It then decides if the offer is accepted or not, or it will provide a counter-offer, as explained in section V-B. When an SLA is signed, a runtime monitor can start monitoring the specific services to assure that the security requirements hold for a specified period of time.

VII. Acknowledgments

This work was partly supported by the EU-funded project TOREADOR [20] (grant no H2020-688797) and by the programme "Piano sostegno alla ricerca 2015-17" funded by the Universita degli Studi di Milano.

References

[1] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, "Big data computing and clouds: Trends and future directions," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 3–15, 2015.

[2] I. Hashem, I. Yaqoob, N. Anuar, S. Mokhtar, A. Gani, and S. Khan, "The rise of big data on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[3] C. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, and C. Hebert, "A model-driven methodology for big data analytics-as-a-service," in *Proc. of IEEE BigData Congress 2017*, Honolulu, HI, USA, June 2017.

[4] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht, "Keystoneml: Optimizing pipelines for large-scale advanced analytics," in *Proc. of ICDE 2017*, San Diego, CA, USA, April 2017.

[5] C. Ardagna, R. Asal, E. Damiani, and Q. Vu, "From security to assurance in the cloud: A survey," *ACM CSUR*, vol. 48, no. 1, pp. 2:1–2:50, August 2015.

[6] M. Anisetti, C. Ardagna, E. Damiani, and F. Saonara, "A test-based security certification scheme for web services," *ACM TWEB*, vol. 7, no. 2, pp. 1–41, May 2013.

[7] M. Krotsiani, G. Spanoudakis, and K. Mahbub, "Incremental certification of cloud services," in *Proc. of SECURWARE 2013*, Barcelona, Spain, August 2013.

[8] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi, "A semi-automatic and trustworthy scheme for continuous cloud service certification," *IEEE TSC*, 2017.

[9] S. Lins, S. Schneider, and A. Sunyaev, "Trust is good, control is better: Creating secure clouds by continuous auditing," *IEEE TCC*, vol. PP, no. 99, pp. 1–1, 2016.

[10] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game," *IEEE TSC*, vol. 11, no. 1, pp. 184–201, 2016.

[11] A. Andrieux, K. Czajkowski, K. Keahey, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (WS-Agreement)," in *Global Grid Forum GRAAP-WG*, Honolulu, HI, USA, June 2004.

[12] K. T. Kearney, F. Torelli, and C. Kotsokalis, "SLA*: An abstract syntax for service level agreements," in *Proc. of Grid 2010*, Brussels, Belgium, October 2010.

[13] R. B. Uriarte, F. Tiezzi, and R. De Nicola, "SLAC: A formal service-level-agreement language for cloud computing," in *Proc. of UCC 2014*, London, UK, December 2014.

[14] Y. Demchenko, P. Membrey, P. Grosso, and C. de Laat, "Addressing big data issues in scientific data infrastructure," in *Proc. of BDDAC 2013*, San Diego, CA, USA, May 2013.

[15] N. Karten, "How to establish service level agreements," 2003.

[16] K. Mahbub and G. Spanoudakis, "Proactive SLA negotiation for service based systems: Initial implementation and evaluation experience," in *Proc. of SCC 2011*, Washington, DC, USA, July 2011.

[17] E. F. Hill, *Jess in Action: Java Rule-Based Systems*. Greenwich, CT, USA: Manning Publications Co., 2003.

[18] G. Norman, D. Parker, and J. Sproston, "Model checking for probabilistic timed automata," *Formal Methods in System Design*, vol. 43, no. 2, pp. 164–190, 2013.

[19] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[20] "Toreador project," http://toreador-project.eu/.