Continuous Architecting of Stream-Based Systems

Marcello Bersani, Francesco Marconi and Damian A. Tamburri
Politecnico di Milano
Milan, Italy
Email: [$marcellomaria.bersani, francesco.marconi,$
$damianandrew.tamburri$]@polimi.it

Pooyan Jamshidi, Andrea Nodari
Imperial College London
London, UK
Email: [$p.jamshidi,$
$a.nodari15$]@imperial.ac.uk

***Abstract*—Big data architectures have been gaining momentum in the last years. For example, Twitter uses stream processing frameworks like Storm to analyse and learn trending topics from billions of tweets per minute. However, maintaining the quality of developed applications (*topologies* in Storm jargon) often requires iterative experiments on expensive multi-node clusters. As an aid to designers and developers evaluating their applications in a DevOps fashion (i.e., through continuous feedback between development and operation), we developed OSTIA, that is, "On-the-fly Static Topology Inference Analysis". OSTIA allows reverse-engineering of topologies for the purpose of: (a) using previously existing verification&validation techniques on elicited models; (b) visualising and refactoring elicited models while maintaining constraints that would only be evaluated at deployment and run-time; (c) tackling the occurrence of common anti-patterns across considered topologies. We illustrate the uses and benefits of OSTIA on three real-life industrial case studies.**

I. Introduction

Big data applications process large amounts of data for the purpose of gaining key business intelligence through complex analytics such as machine-learning [1], [2]. These applications are receiving increased attention in the last years given their ability to yield competitive advantage by direct investigation of user needs and trends hidden in the enormous quantities of data produced daily by the average internet user. Gartner predicts[1] that said business intelligence and analytics applications will remain a top focus for CIOs until at least 2017-2018. However, there are many costs and complexities behind harnessing said applications, ranging from high infrastructure costs to steep learning curves for the different frameworks involved in designing and developing applications for big data, such as Apache Storm[2], Apache Spark[3] or Apache Hadoop[4].

In our own experience with designing and developing for big data, we observed that a key complexity lies in quickly and continuously evaluating the effectiveness of big-data architectures. Effectiveness, in big data terms, means being able to design, deploy, operate, refactor and then (re-)deploy architectures continuously and consistently with runtime restrictions of imposed by frameworks. Storm, for example, requires the processing elements to represent a Directed-Acyclic-Graph (DAG). We argue that this effectiveness can be maintained starting from design time, by enacting a continuous architecting of big-data architectures consistently with a DevOps organisational structure [3], [4]. Such structure eases the (re-)deployability of big data architectures and saves time and effort to run trial-and-error experiments on expensive infrastructure.

To sustain this argument, we developed OSTIA, that stands for: "On-the-fly Static Topology Inference Analysis". OSTIA allows designers and developers to infer the application architecture through on-the-fly reverse-engineering and architecture recovery [5]. During this inference step, OSTIA analyses the architecture to make sure it is consistent with restrictions and constraints of the big data frameworks at hand. Also, in an effort to tackle said complexities and offer support in a DevOps fashion, OSTIA was engineered to act as a mechanism that closes the feedback loop between operating the frameworks (-Ops phase) and their development and improvement phase (Dev- phase).

Currently, OSTIA focuses on Storm, i.e., one of the most famous and established real-time stream processing big data engine [6], [7]. The core element of Storm, is called *topology*, which represents the architecture of the processing components of the application (from now we use topology and architecture interchangeably).

OSTIA hardcodes intimate knowledge on the streaming development framework (Storm, in our case) and its dependence structure in the form of a meta-model [8]. This knowledge is necessary to make sure that elicited topologies are correct, so that models may be used in at least four scenarios: (a) realising an exportable visual representation for said topologies; (b) checking said topologies against framework restrictions that would only become evident during infrastructure setup or run-time operation; (c) checking that said topologies do not show any anti-patterns [9] that may lower performance and limit deployability/executability; (d) finally, use said topologies for further analysis, e.g., through model verification [10].

This paper outlines OSTIA, elaborating its major usage scenarios and its benefits while discussing and addressing its limitations. Also, we evaluate OSTIA using industrial case-study research featuring an open-source social-sensing application to show that OSTIA yields valuable design and development insights using an inference analysis of the three static topologies behind said industrial application. Finally, we elaborate on how OSTIA helps the continuous architecting of

[1] http://www.gartner.com/newsroom/id/2637615
[2] http://storm.apache.org/
[3] http://spark.apache.org/
[4] https://hadoop.apache.org/

recovered topologies by means of valuable on-the-fly analyses as well as ancillary verification techniques. We conclude that OSTIA provides valuable insights for software developers to continuously architect and increase quality of their big-data design and development featuring Storm, so that expedite (re-)deployment can take place.

The rest of the paper is structured as follows. Section II outlines our research problem, research questions and our approach at tackling them. Sections III, IV and V describes OSTIA, discussing the usage scenarios and (anti-)patterns it was designed to support, while listing the main benefits we perceived in using it. Section VI elaborates further on the benefits by providing an actual evaluation of OSTIA using three cases from industrial practice. Section VII discusses the results and evaluation, also outlining OSTIA limitations and potential threats to its validity. Finally, Sections VIII and IX report related work and conclude the paper.

II. Research Design

The work we elaborated in this paper is stemming from the following research question:

"Can we assist the continuous architecting of stream-based systems design?"

The results contained in this paper in response to this research question were initially elaborated within a free-form focus group [11] involving three experienced practitioners and researchers on big data streaming technologies, such as Storm. Following the focus group, through self-ethnography [12] and brainstorming we identified the series of essential consistency checks, algorithmic evaluations as well as anti-patterns that can now be applied through OSTIA while recovering an architectural representation for Storm topologies. We designed OSTIA[5] to support the incremental and iterative refinement of streaming topologies based on the incremental discovery and correction of the above checks and patterns.

In addition, we incrementally refined and evaluated OSTIA through scenario-analysis and case-study research [13] involving an industrial partner.

As previously stated, OSTIA was evaluated by means of an industrial case-study offered by one of the industrial partners in the DICE EU H2020 Project consortium[6]. The partner in question uses open-source social-sensing software to elaborate a subscription-based big-data application that: (a) aggregates news assets from various sources (e.g., Twitter, Facebook, etc.) based on user-desired specifications (e.g., topic, sentiment, etc.); (b) presents and allows the manipulation of data. Said application is based on the SocialSensor App[7] which features the combined action of three complex streaming topologies based on Apache Storm (see Fig. 1 for a sample of topologies).

In particular, the topology in Fig. 1 extracts data from sources and manipulates said data to divide and arrange contents based on type (e.g., article vs. media), later updating a series of databases (e.g., Redis) with these elaborations.

The models that OSTIA elicited from this application were showcased to our industrial partner in a focus group. Our qualitative analysis was based on questionnaires and open discussion.

Finally, we applied well-established verification approaches to integrate the value and benefits behind using OSTIA. We engineered OSTIA to support exporting of elicited topologies for their further analysis using the Zot LTL model-checker [14] using the approach outlined in our previous work [10].

III. Research Solution

This section outlines OSTIA starting form a brief recap of the technology it is currently designed to support, i.e., the Apache Storm framework. Further on, the section introduces how OSTIA was designed to support continuous architecting of streaming topologies focusing on Storm. Finally, the section outlines the meta-model for Storm that captures all restrictions and rules (e.g., for configuration, topology, dependence, messaging, etc.) in the framework. OSTIA uses this meta-model as a reference every time the application is run to recover and analyse operational topologies.

A. Storm architecture

Storm is a technology developed at Twitter [7] in order to face the problem of processing of streaming of data. It is defined as a distributed processing framework which is able to analyse streams of data. The core element in the system is called *topology*. A Storm topology is a computational graph composed by nodes of two types: spouts and bolts. The former type includes nodes that process the data entering the topology, for instance querying APIs or retrieve information from a message broker, such as Apache Kafka. The latter executes operations on data, such as filtering or serialising.

B. OSTIA design

The overall architecture of OSTIA is depicted in Figure **??**. The logical architectural information of the topology is retrieved by OSTIA via static analysis of the source code. OSTIA generates a simple intermediate format to be used by other algorithmic processes.

OSTIA is architected in a way that algorithmic analysis, such as anti-pattern analyses, can be easily added. These analyses use the information resides in the intermediate format and provide added value analyses for continuous architecting of storm topologies. Since the information in the intermediate format only rely on the logical code analysis, some algorithmic analyses requires some information regarding the running topology, such as end to end latency and throughput.

Such information will be continuously added to the intermediate repository via runtime monitoring of the topology on real deployment cluster. These provide appropriate and rich information for refactoring the initial architecture and enabling performance driven DevOps [15].

[5] https://github.com/maelstromdat/OSTIA
[6] http://www.dice-h2020.eu/
[7] https://github.com/socialsensor

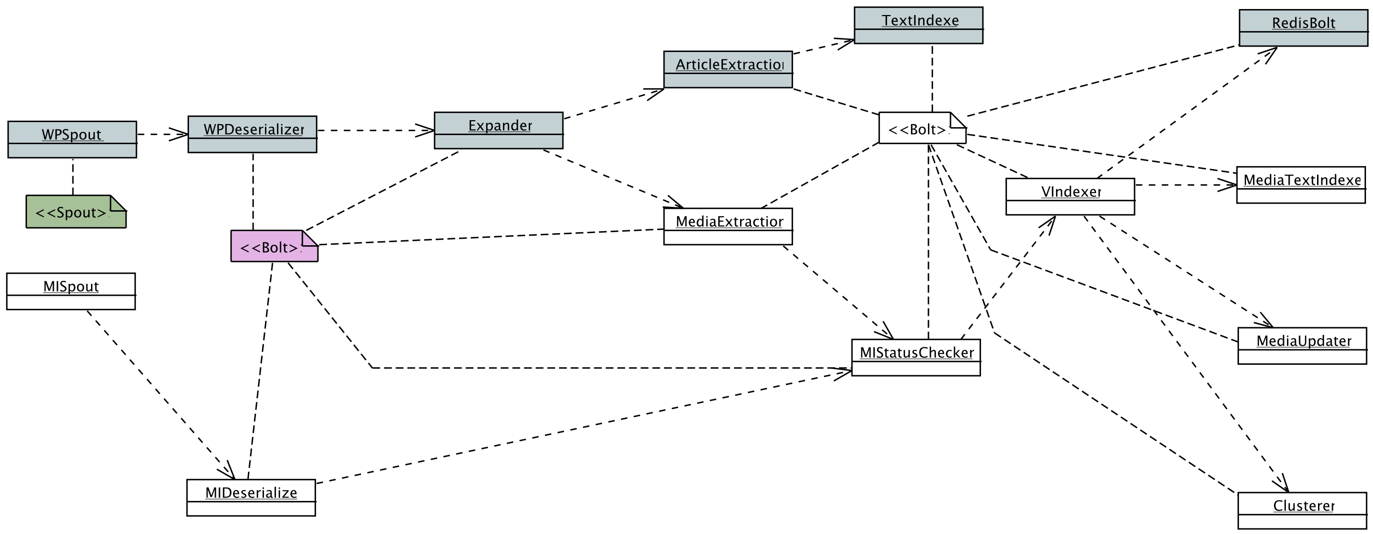

Fig. 1. A sample Storm topology in the SocialSensor App.

Finally, OSTIA allows users to export the topology in different formats (e.g. JSON) to analyse the topology with other tools.

C. Deep Within OSTIA: The Storm Framework Meta-Model

OSTIA was designed to retrieve and analyse Storm topologies on-the-fly, allowing their refactoring in a way which is consistent with framework restrictions, rules and regulations part of the Storm framework. In so doing, OSTIA uses a meta-model for the Storm framework. This meta-model acts as an image of all said restrictions and rules that OSTIA needs to maintain for elicited (or refactored) topologies. Essentially OSTIA implements in the meta-model the operational picture of the Storm Framework - this is critical to checking that the restrictions and constraints coded within Storm are reflected in elicited models as well as maintained during architecting and refactoring of those models. In addition, OSTIA applies all the necessary anti-pattern checks in combination with checking that said framework restrictions are maintained - this is critical to support continuous architecting in a manner which is consistent with Storm restrictions that would only become apparent during run-time and operations. The meta-model in question is depicted in Fig. 3. The figure shows an overview of the meta-model for Storm[8] where, for example, the grouping restrictions that Storm envisions are captured in an enumeration of constraints (see the <<Grouping>> element or the <<ReplicationFactor>> concrete parameter). Key elements of the meta-model are the following:

- the <<TopologyConfiguration>> meta-element contains the parameters necessary for the Storm framework to be configured and to run on the selected infrastructure. OSTIA checks that these parameters are present or that defaults are correctly inplace;
- the <<TopologyConfiguration>> meta-element specifies the topological construct being elicited for the analysed Storm application, as composed of the <<Bolt>> and the <<Spout>> meta-elements;
- the <<Grouping>> meta-element contains restrictions on the possible groupings of the <<Bolt>> and the <<Spout>> meta-elements within the elicited topology. OSTIA uses these restrictions to analyse the elicited topology;

[8]The details of this meta-model and the restrictions captured therein is beyond the scope of this paper. More details are available in our project homepage, in section D2.1: http://dice-h2020.eu/deliverables/D2.1.

IV. Topology Design Anti-Patterns

This section and section V elaborate on the Anti-patterns and algorithmic analysis supported in OSTIA. All figures in these sections use a simple graph-like notation where nodes may be any topological element (e.g., Spouts or Bolts in Apache Storm terms) while edges are to be interpreted as directed data-flow connections.

This elaborates on the anti-patterns we elicited through self-ethnography. These anti-patterns are elaborated further within OSTIA to allow for their detection during streaming topology inference analysis.

A. Multi-Anchoring

In order to guarantee fault-tolerant stream processing, tuples processed by bolts needs to be anchored with the unique id of the bolt and be passed to multiple acknowledgers (or "ackers" in short) in the topology. In this way, ackers can keep track of tuples in the topology.

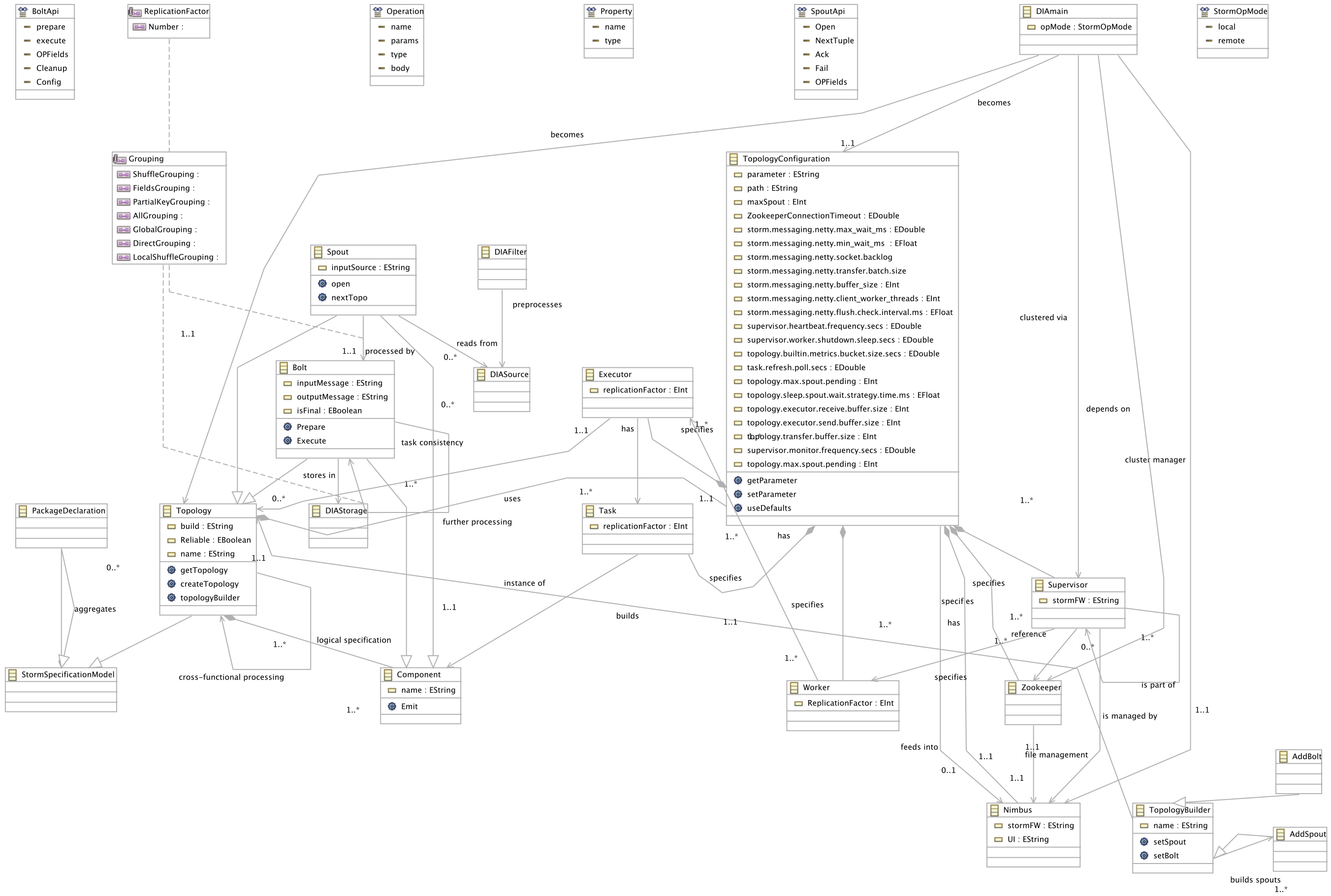

Fig. 3. Deep Within OSTIA, a Storm Meta-Model.

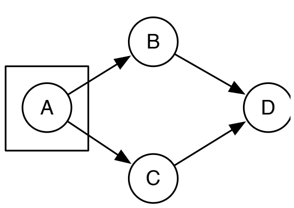

Fig. 4. Multi-anchoring.

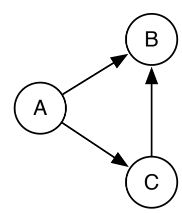

Fig. 5. Cycle-in Topology.

B. Cycle-in Topology

Technically, it is possible to have cycle in Storm topologies. An infinite cycle of processing would create an infinite tuple tree and make it impossible for Storm to ever acknowledge spout emitted tuples. Therefore, cycles should be avoided or resulting tuple trees should be investigated additionally to make sure they terminate at some point and under a specified series of conditions. The anti-pattern itself may lead to infrastructure overloading and therefore increased infrastructure costs.

C. Persistent Data

This pattern covers the circumstance wherefore if two processing elements need to update a same entity in a storage, there should be a consistency mechanism in place. OSTIA offers limited support to this feature, which we plan to look into more carefully for future work. More details on this support are discussed in the approach limitations section (see Sec. VII-B).

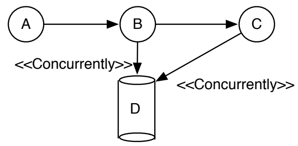

Fig. 6. Concurrency management.

V. Algorithmic Analysis on Stream Topologies

A. fan-in/fan-out

For each element of the topology, fan-in is the number of incoming streams. Conversely, fan-out is the number outgoing streams. In the case of bolts, both in and out streams are internal to the topology. For Spouts, incoming streams are the data sources of the topology (e.g., message brokers, APIs, etc).

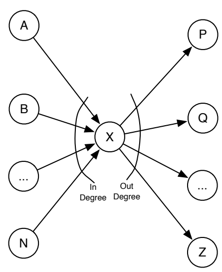

Fig. 7. Fan-in fan-out in Stream topologies.

B. topology cascading

By topology cascading, we mean connecting two different Storm topologies via a messaging framework (e.g., Apache Kafka). This circumstance, which is actually part of our evaluation and case-studies, may raise the complexity of the overarching topology to unacceptable levels and may require additional attention. OSTIA support for this feature is still limited, more details on this and similar limitations are discussed in Section VII-B.

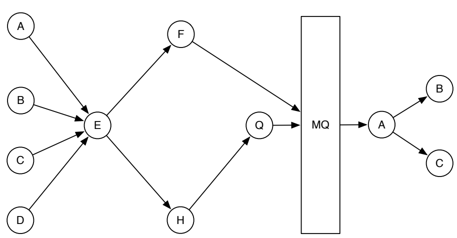

Fig. 8. cascading.

C. Topology clustering

Identifying the coupled processing elements and put the in a cluster in a way that elements in a cluster have high cohesion and less coupled with the elements in other clusters. Simple clustering or Social-Network Analysis mechanisms can be used to infer clusters. These clusters may require additional attention since they could turn out to become bottlenecks. Reasoning more deeply on clusters and their resolution may lead to establishing the Storm scheduling policy best-fitting with the application at hand.

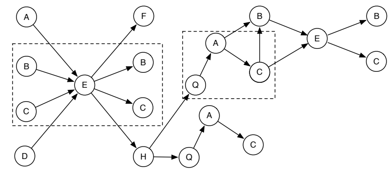

Fig. 9. clustering.

D. Computation funnel

A computational funnel emerges when there is not a path from data source (spout) to the bolts that sends out the tuples off the topology to another topology through a messaging framework or through a storage. This circumstance should be dealt with since it may compromise the availability of results under the desired performance restrictions.

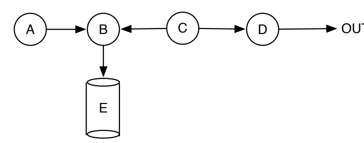

Fig. 10. computation funnel.

E. Linearising a topology

Sorting the processing elements in a topology in a way that topology looks more linear, visually. This step ensures that visual investigation and evaluation of the structural complexity of the topology is possible by direct observation. It is sometimes essential to provide such a visualisation to evaluate how to refactor the topology based on emerging needs.

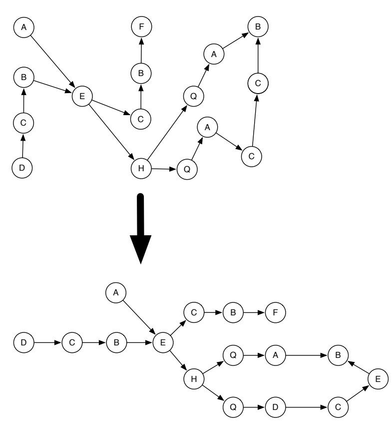

Fig. 11. linearizing.

VI. EVALUATION

¡¡¡¡¡¡¡ HEAD
======= ¿¿¿¿¿¿¿ 915d87a8f9df7f67cfd959767461c7f00a2f4d2b

¡¡¡¡¡¡¡ HEAD
This section describes the formal modeling and validation employed in OSTIA which both rely on *satisfiability checking* [16], an alternative approach to model-checking. Instead of an operational model (like automata or transition systems), as in model-checking, the system (i.e., a topology in this context) is specified by a formula defining their executions over time and properties are verified by proving that the system logically entails them.

The logic we use is Constraint LTL over clocks (CLTLoc) [18] which is a semantic restriction of Constraint LTL (CLTL) [17] allowing atomic formulae over $(\mathbb{R}, \{<,=\})$ where the arithmetical variables behave like clocks of Timed Automata (TA) [**?**]. A clock x measures the time elapsed since the last time when $x = 0$ held, i.e., since the last "reset" of x. Clocks are interpreted over Reals and their value can be tested with respect to a positive integer value. Let X be a finite set of clock variables x over $\mathbb{R}$ and AP be a finite set of atomic propositions p. CLTLoc formulae are defined as follows:

$$\phi := \ p \mid x \sim c \mid \phi \wedge \phi \mid \neg\phi \mid \mathbf{X}(\phi) \mid \mathbf{Y}(\phi) \mid \phi\mathbf{U}\phi \mid \phi\mathbf{S}\phi$$

where $c \in \mathbb{N}$ and $\sim \in \{<,=\}$, $\bullet$, $\circ$, $\mathbf{U}$ and $\mathbf{S}$ are the usual "next", "previous", "until" and "since". A *model* is a pair (π, σ), where σ is a mapping associating every variable x and position in $\mathbb{N}$ with value $\sigma(i, x)$ and π is a mapping associating each position in $\mathbb{N}$ with subset of AP. The semantics of CLTLoc is defined as for LTL except for formulae $x \sim c$. At position $i \in \mathbb{N}$, $(\pi, \sigma), i \models x \sim c$ **iff** $\sigma(i, x) \sim c$. A formula is *satisfiable* if it has a model.

The standard technique to prove the satisfiability of CLTL and CLTLoc formulae is based on of Büchi automata [17], [18] but, for practical implementation, Bounded Satisfiability Checking (BSC) [16] avoids the onerous construction of automata. By unrolling the semantics of a formula for a finite number $k > 0$ of steps, the outcome of a BSC problem is either an infinite ultimately periodic model or unsat. [18] shows that BSC for CLTLoc is complete and that is reducible to a decidable Satisfiability Modulo Theory (SMT) problem. A CLTLoc formula can be translated into the decidable theory of quantifier-free formulae with equality and uninterpreted functions combined with the theory of Reals over $(\mathbb{R}, <)$.

CLTLoc allows the specification of temporal constraints using clock variables ranging over $\mathbb{R}$, whose value is not abstracted. Clock variables represent, in the logical language and with the same precision, physical (dense) clocks. They appear in formulae of the form $x \sim c$ to express a bound c on the delay measured by clock x. Clocks are associated with specific events to measure time elapsing over the execution. As they are reset when the associated event occurs, in any moment, the clock value represents the time elapsed since the previous reset and corresponds to the elapsed time since the last occurrence of the event associated to it. We use such constraints to define, for instance, the time delay required to process tuples or between two node failures.

Modeling topologies requires to express by formulae emitting rates which measure the number of tuples emitted by a spout node per time unit. ***TBC

VII. DISCUSSION

This section discusses our findings, their value in improving the quality of big-data architectures as well as the insights we discovered to plan further improvements of our technology. Finally, this section discusses the limitations behind using OSTIA.

A. Findings and Quality-Improvement Insights

OSTIA represents one humble, but significant step at supporting practically the necessities behind developing and maintaining high-quality big-data applications. In designing and developing OSTIA we encountered a number of insights that may lead to improving said quality by means of OSTIA.

First, we found that it is often more useful to develop a quick-and-dirty but "runnable" topology than improving the topology at design time for a tentatively perfect execution. This way of working is consistent with continuous improvement approaches typical in DevOps scenarios. Also, this is mostly the case with big-data applications that are developed stemming from previously existing topologies. OSTIA hardcodes this way of thinking by supporting reverse-engineering and recovery of deployed topologies for their incremental improvement. Although we did not carry out extensive qualitative or quantitative evaluation of OSTIA in this regard, we are planning additional industrial experiments for future work with the goal of increasing OSTIA usability and practical quality.

Second, big-data applications design is an extremely young and emerging field for which not many software design patterns have been discovered yet. The (anti-)patterns and approaches currently hardcoded into OSTIA are inherited from related fields, e.g., pattern- and cluster-based graph analysis. Nevertheless, OSTIA may also be used to investigate the existence of recurrent and effective design solutions (i.e., design patterns) for the benefit of big-data application design. We are planning to improve OSTIA in this regard by experimenting on two fronts: (a) re-design and extend the facilities with which OSTIA supports anti-pattern detection; (b) run OSTIA on multiple big-data applications stemming from multiple technologies beyond Storm (e.g., Spark, Hadoop Map Reduce, etc.) with the purpose of finding recurrent patterns.

Third, finally, a step which is currently unsupported during big-data applications design is devising an efficient algorithmic breakdown of a workflow into an efficient topology. However, OSTIA does support the linearisation and combination of multiple topologies, e.g., into a cascade. Cascading and similar super-structures may be an interesting investigation venue since they may reveal more efficient styles for big-data architectures beyond styles such as Lambda Architecture [19]. OSTIA may aid in this investigation by allowing the interactive and incremental improvement of multiple (combinations of) topologies together.

B. Approach Limitations and Threats to Validity

Although OSTIA shows promise both conceptually and as a practical tool, it shows several limitations by design in its current form.

First of all, OSTIA only supports streaming topologies enacted using Storm. Multiple other big-data frameworks exist, however, to support both streaming and batch processing.

Second, OSTIA only allows to recover and evaluate previously-existing topologies, its usage is limited to design improvement and refactoring phases rather than design. Although this limitation may inhibit practitioners from using our technology, the (anti-)patterns and algorithmic approaches elaborated in this paper help designers and implementors to develop the reasonably good-quality and "quick" topologies upon which to use OSTIA for continuous improvement.

Third, OSTIA does offer essential insights to aid deployment as well (e.g., separating or *clustering* complex portions of a topology so that they may run on dedicated infrastructure), however, our tool was not meant to be used as a system to aid planning and infrastructure design. Rather, as specified previously in the introduction, OSTIA was meant to evaluate and increase the quality of topologies *before* they enter into operation since the continuous improvement cycles connected to operating the topology and learning from said operation are often costly and still greatly inefficient.

Fourth, although we were able to discover a number of recurrent anti-patterns to be applied during OSTIA analysis, we were not able to implement all of them in practice and in a manner which allows to spot both the anti-pattern and any problems connected with it. For example, detecting the "Cycle-in topology" is already possible, however, OSTIA would not allow designers to understand the consequence of the anti-pattern, i.e., where in the infrastructure do the cycles cause troubles. Also, there are several features that are currently implemented but not working within OSTIA, for example, the "Persistent Data" and the "Topology Cascading" features.

In the future we plan to tackle the above limitations furthering our understanding of streaming design as well as the support OSTIA offers to designers during continuous architecting.

VIII. Previous and Related Work

The work behind OSTIA stems from the EU H2020 Project called DICE[9] where we are investigating the use of model-driven facilities to support the design and quality enhancement of Big-Data applications. In the context of DICE, much previous work has been done to support the design, development and deployment of Big-Data applications. For example, we have been developing a series of ad-hoc technological specifications, i.e., meta-model packages that contain all concepts, constructs and constraints needed to develop and operate Big-Data applications for the frameworks coded into the technological specifications. These specifications can be used, for example, to instantiate Big-Data components following standard Model-Driven procedures, without the need to learn said frameworks at all. OSTIA uses insights gained in developing and using said frameworks to apply consistency checks in the context of recovering Big-Data architectures, specifically, for Storm. Much similarly to the DICE effort, the IBM Stream Processing Language (SPL) initiative [20] provides an implementation language specific to programming streams management (e.g., Storm jobs) and related reactive systems based on the Big-Data paradigm. Although SPL is specific to WebSphere and IBM technology, its attempt at providing a relatively abstract language to implement for streams management and processing is remarkably related to OSTIA, since one of its aims is to improve quality of streams management by direct codification of higher order concepts such as streams declarations.

In addition, there are several work close to OSTIA in terms of their foundations and type of support.

First, from a quantitative perspective, much literature discusses quality analyses of Storm topologies, e.g., from a performance [21] or reliability point of view [22]. Said works use complex math-based approaches to evaluating a number of Big data architectures, their structure and general configuration. However, although novel, these approaches do not suggest any significant design improvement method or pattern to make the improvements *deployable*. With OSTIA, we make available a tool that automatically elicits a Storm topology and, while doing so, analyses said topology to evaluate it against a number of consistency checks that make the topology consistent with the framework it was developed for (Storm, in our case). As previously introduced, a very trivial example of said checks consists in evaluating wether the topology is indeed a Directed-Acyclic-Graph (DAG), as per constraints of the Storm framework. To the best of our knowledge, no such tool exists to date.

Second, in our previous work, we proposed BO4CO [23], an approach for locating optimal configurations using ideas of carefully choosing where to sample by sequentially reducing uncertainty in the response surface approximation in order to reduce the number of performance measurements. We have carried out extensive experiments with three different stream topologies running on Apache Storm. Experimental results demonstrate that BO4CO outperforms the baselines in terms of distance to the optimum performance with at least an order of magnitude.

Third, from a modelling perspective, approaches such as StormGen [24] offer means to develop Storm topologies in a model-driven fashion using a combination of generative techniques based on XText and heavyweight (meta-)modelling, based on EMF, the standard Eclipse Modelling Framework Format. Although the first of its kind, StormGen merely allows the specification of a Storm topology, without applying any consistency checks or without offering the possibility

[9] http://www.dice-h2020.eu/

to *recover* said topology once it has been developed. By means of OSTIA, designers and developers can work hand in hand while refining their Storm topologies, e.g., as a consequence of verification or failed checks through OSTIA. As a consequence, tools such as StormGen can be used to assist the preliminary development of quick-and-dirty Storm topologies, e.g., to be further processed and improved with the aid of OSTIA.

Fourth, from a verification perspective, ...

@Marcello,Francesco: here we should probably elaborate on what kind of verification approach we are using and what other verifications may be done, e.g., using some related work at this point... e.g., is there any other verification attempt considering JSON as an interchange format? I would discuss these and compare them to OSTIA as a whole

Finally, several deployment modelling technologies may be related to OSTIA since their role is to model the deployment structure represented by Big data architectures so that it can actually be deployed using compliant orchestrators. One such example is Celar[10], a deployment modelling technology based on the TOSCA OASIS Standard[11]. Celar and related technologies (e.g., Alien4Cloud[12]) may be used in combination with OSTIA since their role is that of representing the infrastructure needed by modelled (Big data) applications so that they can be deployed. This representation is realised by means of infrastructure blueprints to be run by compliant orchestrators. The role of OSTIA in this scenario, is that of helping the quality refinement of an application topology to represent the very infrastructure needed for its run-time environment.

IX. Conclusion

Big data applications are rapidly gaining interest and momentum by small to big players on the market, even beyond the IT corner. Applications that make heavy use of Big data application frameworks require intensive reasoning and continuous architecting of design aspects typically around the topology of the basic operations to be applied in manipulating the data.

We set out to answer the following research question: "Can we assist the continuous architecting of Big-Data streaming designs?" As an answer to this question this paper elaborates on OSTIA, a toolkit to assist designers and developers in this continuous architecting campaign. OSTIA helps designers and developers by recovering and analysing the architectural topology on-the-fly, assisting them in: (a) reasoning on the topological structure and how to refine it; (b) export the topological structure consistently with restrictions of their reference development framework so that further analysis (e.g., formal verification) may ensue. In addition, while performing on-the-fly architecture recovery, the analyses that OSTIA is able to apply focus on checking for the compliance to essential consistency rules specific to targeted big data frameworks. Finally, OSTIA allows to check wether the recovered topologies contain occurrences of key anti-patterns we elicited and studied in our own experience and previous work.

By running a case-study with a partner organization, we observed that OSTIA assists designers and developers in establishing and continuously improving the quality of topologies behind their big data applications in multiple ways.

In the future we plan to further our understanding of the anti-patterns that may emerge across big data topologies, e.g., as discussed, by learning said anti-patterns by using graphs analysis techniques inherited from social-networks analysis. Also, we plan to expand OSTIA to support further technologies beyond the most common application framework for streaming, i.e., Storm. Finally, we plan to further evaluate OSTIA using more ad-hoc empirical evaluation campaigns in industry.

[10]https://github.com/CELAR/c-Eclipse

[11]https://www.oasis-open.org/apps/org/workgroup/tosca/

[12]http://alien4cloud.github.io/

Acknowledgment

The Authors' work is partially supported by the European Commission grant no. 644869 (EU H2020), DICE. Also, Damian's work is partially supported by the European Commission grant no. 610531 (FP7 ICT Call 10), SeaClouds.

References

[1] C. K. Emani, N. Cullot, and C. Nicolle, "Understandable big data: A survey." *Computer Science Review*, vol. 17, pp. 70–81, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/csr/csr17.html#EmaniCN15

[2] B. Ratner, *Statistical and Machine-Learning Data Mining: Techniques for Better Predictive Modeling and Analysis of Big Data*. CRC PressINC, 2012. [Online]. Available: http://books.google.de/books?id=0KuIJTAznLUC

[3] D. A. Tamburri, P. Lago, and H. van Vliet, "Organizational social structures for software engineering." *ACM Comput. Surv.*, vol. 46, no. 1, p. 3, 2013.

[4] L. J. Bass, I. M. Weber, and L. Zhu, *DevOps - A Software Architect's Perspective.*, ser. SEI series in software engineering. Addison-Wesley, 2015.

[5] I. Ivkovic and M. W. Godfrey, "Enhancing domain-specific software architecture recovery." in *IWPC*. IEEE Computer Society, 2003, pp. 266–273. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwpc/iwpc2003.html#IvkovicG03

[6] R. Evans, "Apache storm, a hands on tutorial." in *IC2E*. IEEE, 2015, p. 2. [Online]. Available: http://dblp.uni-trier.de/db/conf/ic2e/ic2e2015.html#Evans15

[7] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[8] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[9] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-oriented software architecture. Vol. 2, Patterns for concurrent and networked objects*. John Wiley, Chichester, England, 2000.

[10] M. M. Bersani, S. Distefano, L. Ferrucci, and M. Mazzara, "A timed semantics of workflows." in *ICSOFT (Selected Papers)*, ser. Communications in Computer and Information Science, A. Holzinger, J. Cardoso, J. Cordeiro, T. Libourel, L. A. Maciaszek, and M. van Sinderen, Eds., vol. 555. Springer, 2014, pp. 365–383. [Online]. Available: http://dblp.uni-trier.de/db/conf/icsoft/icsoft2014s.html#BersaniDFM14

[11] D. L. Morgan, *Focus groups as qualitative research*. Sage Publications, Thousand Oaks, 1997.

[12] M. Hammersley and P. Atkinson, *Ethnography*. London: Routledge, 2003.

[13] S. B. Merriam, *Case study research in education: A qualitative approach*. Jossey-Bass Publishers, 1991, vol. 1, no. 1980. [Online]. Available: http://psycnet.apa.org/psycinfo/1989-97006-000

[14] C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi, "Modeling time in computing: A taxonomy and a comparative survey," *ACM Comput. Surv.*, vol. 42, no. 2, pp. 6:1–6:59, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1667062.1667063

[15] A. Brunnert, A. van Hoorn, F. Willnecker, A. Danciu, W. Hasselbring, C. Heger, N. Herbst, P. Jamshidi, R. Jung, J. von Kistowski *et al.*, "Performance-oriented devops: A research agenda," *arXiv preprint arXiv:1508.04752*, 2015.

[16] M. Pradella, A. Morzenti, and P. S. Pietro, "Bounded satisfiability checking of metric temporal logic specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 20:1–20:54, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491509.2491514

[17] S. Demri and D. D'Souza, "An automata-theoretic approach to constraint LTL," *Information and Computation*, vol. 205, no. 3, pp. 380–415, 2007.

[18] M. M. Bersani, M. Rossi, and P. San Pietro, "A tool for deciding the satisfiability of continuous-time metric temporal logic," *Acta Informatica*, pp. 1–36, 2015.

[19] M. Quartulli, J. Lozano, and I. G. Olaizola, "Beyond the lambda architecture: Effective scheduling for large scale eo information mining and interactive thematic mapping." in *IGARSS*. IEEE, 2015, pp. 1492–1495. [Online]. Available: http://dblp.uni-trier.de/db/conf/igarss/igarss2015.html#QuartulliLO15

[20] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. P. Mendell, H. Nasgaard, S. Schneider, R. Soul, and K.-L. Wu, "Ibm streams processing language: Analyzing big data in motion." *IBM Journal of Research and Development*, vol. 57, no. 3/4, p. 7, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/ibmrd/ibmrd57.html#HirzelAGJKKMNSSW13

[21] D. Wang and J. Liu, "Optimizing big data processing performance in the public cloud: opportunities and approaches." *IEEE Network*, vol. 29, no. 5, pp. 31–35, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/network/network29.html#WangL15

[22] Y. Tamura and S. Yamada, "Reliability analysis based on a jump diffusion model with two wiener processes for cloud computing with big data." *Entropy*, vol. 17, no. 7, pp. 4533–4546, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/entropy/entropy17.html#TamuraY15

[23] P. Jamshidi and G. Casale, "An uncertainty-aware approach to optimal configuration of stream processing systems." in *VLDB (under evaluation)*, 2016.

[24] K. Chandrasekaran, S. Santurkar, and A. Arora, "Stormgen - a domain specific language to create ad-hoc storm topologies." in *FedCSIS*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2014, pp. 1621–1628. [Online]. Available: http://dblp.uni-trier.de/db/conf/fedcsis/fedcsis2014.html#ChandrasekaranSA14