Trace Checking of Streaming Applications through DICE-TraCT

Marcello M. Bersani
Politecnico di Milano
Milan, Italy
marcellomaria.bersani@polimi.it

Francesco Marconi
Politecnico di Milano
Milan, Italy
francesco.marconi@polimi.it

Matteo Rossi
Politecnico di Milano
Milan, Italy
matteo.rossi@polimi.it

ABSTRACT

This paper introduces DICE-TraCT, the tool—part of the DICE toolchain—that allows developers of Data Intensive Applications to analyze traces of executions of such applications and detect deviations from the expected behavior. The tool works in tandem with the companion formal verification tool D-VerT, to check that the parameters used for the sizing of applications and that guarantee the desired safety and timing properties are indeed correct.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**;

KEYWORDS

Trace-checking; Data Intensive Applications; Storm Topologies

ACM Reference Format:
Marcello M. Bersani, Francesco Marconi, and Matteo Rossi. 2018. Trace Checking of Streaming Applications through DICE-TraCT. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3185768.3186287

1 INTRODUCTION

Trace checking is an approach for the analysis of system executions that are recorded as sequences of timestamped events. The purpose of the analysis is to establish whether the system logs satisfy a property, usually specified in a logical language. Trace checking is especially useful when the aggregated data, available from the monitoring system of the application, are not enough to derive certain metric values that have to be calculated with respect to some specific criteria. In some cases, in fact, the criteria are application-dependent as they are related to some non-functional property of the application. Trace checking can be employed to achieve an application-dependent log analysis and to extract specific information from the executions of a running application.

In this paper we introduce the DICE Trace Checking Tool (DICE-TraCT), which is part of the DICE toolchain. The tool allows developers of Data Intensive Applications (DIAs) to analyze traces of executions of such applications and detect deviations from the expected behavior. The paper first provides an overview of the tool in Section 2, then shows usage of the tool on a use case (Section 3).

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186287

2 TOOL OVERVIEW

According to the DICE [2] vision, trace checking is performed after verification to allow for continuous model refinement. The verification task in DICE [1] has the goal of certifying the correctness of the applications in terms of properties that are specific for the target technology implementing the application. The result obtained through the log analysis is used to confirm or refute the outcome of the verification task, which is run at design time. The value of the parameters in the design-time model is compared with the value at runtime; if the two are "compatible" then the results of verification are valid, otherwise the model must be refined.

Usage scenario. DICE-TraCT has been developed to perform log analysis of Storm applications. The log traces collected from the monitoring platform are analyzed to certify the adherence to the behavioral model that is assumed at design time. If the runtime behavior does not conform with the application design, then the design must be refined and later verified to obtain a new certification of correctness. Trace-checking can be applied to provide information to D-VerT [3], the tool that enables the formal verification of Storm topologies. Verification of Storm topologies takes place on UML models enriched with information that represents the application behavior over time. Trace checking extracts from real executions the parameter values of the model used by D-VerT that are not available from the monitoring service of the framework, as they are inherently specific of the modeling adopted for the verification—for example, the ratio between the number of messages that are received by a bolt and the number of messages that it emits in output.

Tool Architecture. DICE-TraCT collects analysis requests from the DICE-IDE that are issued by the users and, based on the information retrieved through the queries sent to the monitoring platform, executes one or more instances of trace checking. The DICE-IDE allows the user to select a property to verify for the selected application, currently shown in the IDE, and run the trace checking. The input format for DICE-TraCT is a JSON file which contains the name of the topology to verify, the set of nodes that the user wants to analyze and the property to verify. The current version of the tool does not support user-defined properties, but only those related to parameter of the verification model. Figure 1 shows the architecture of DICE-TraCT.

Trace Checking Engine (TCE) performs the trace analysis. The output is a Boolean outcome which is the result of the evaluation of the formula over the specified log. The positive outcome is obtained if the log satisfies the property.

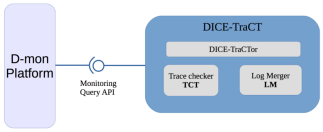

Figure 1: DICE-TraCT architecure.

Storm computations are realized through workers that concurrently run various executors over distinct machines of a cluster of physical nodes. Worker logs might contain more than one sequence of events, each one associated with an executor spawned in that worker. A topology node, either spout or bolt, might then be deployed over different workers and the information related to a single node, either spout or bolt, may be spread over many log files. However, TCE can analyze one log file at a time. Hence, DICE-TraCT elaborates the logs and it aggregates all the events related to a node (or a subset of nodes) into a new log trace. **Log Merger** (LM) receives in input a set of worker logs of a deployed topology under monitoring and a description of the topology listing all its computational nodes. The outcome it produces is a set of logs where each log records all, and only, the events related to a certain node in the topology.

DICE-TraCTor coordinates the activity of LM and TCE upon a request from the DICE-IDE. First, it builds the inputs file to run LM, runs suitable transformations on the new extracted logs, if they are needed to run trace checking, and then defines the input file of the property for TCE. Finally, it runs TCE and, when TCE terminates, it notifies the outcome of the analysis.

DICE-TraCT can be accessed through a POST call with the following input data: (i) parameters *ip* and *port* specify the IP address and the port where the DICE-TraCT service is deployed and running; (ii) the payload is a json descriptor which defines the trace-checking analysis to be carried out.

DICE-TraCT is implemented by a Flask (http://flask.pocoo.org/) module `dicetractservice.py`. The function `dicetract()`, which is associated with the POST method `/run`, implements the DICE-TraCT functionalities. Assuming that DICE-TraCT is deployed at *dicetracturl* on port 5050 and that the monitoring platform is deployed at *dmonurl* at port *dmonport*, an example of a method call can be the following: `POST http://dicetracturl:5050/run?ip=dmonurl&port=dmonport`.

The payload associated with the POST call is a descriptor which defines the parameters to run trace checking for a given topology. The descriptor is a JSON file that is built through the DICE-IDE by the user who monitors the topology. The information stored into JSON fields are the following: *a)* a field that specifies the topology name which is used by DICE-TraCT to query the monitoring platform and obtain all the necessary log files to perform trace checking. *b)* A list of descriptors that specify, for each node, a non-functional property to check. The properties can be related to parameters of the verification model like, for instance, the ratio *sigma* between the number of messages in input and the number of messages in output of a Storm node; or any user-defined property which can be translated by DICE-TraCT into a trace checking instance. *c)* A list of formulae descriptors that specify user-defined

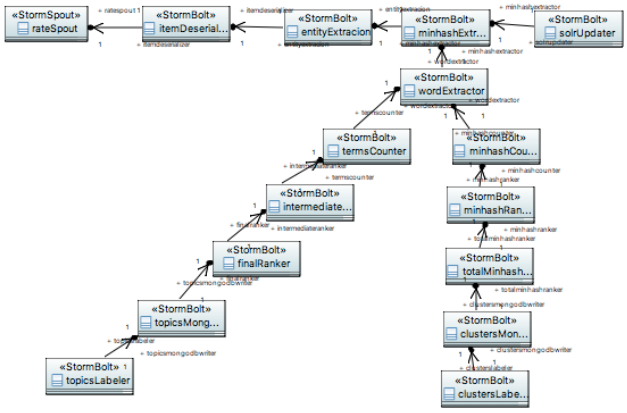

Figure 2: Trend Topic Topology.

logical formulae to be used for trace-checking. DICE-TraCT exploits a direct connection to the monitoring platform of the DICE framework which is used to retrieve and collect log traces of the application currently analyzed. By means of suitable API methods, DICE-TraCT can: (i) activate a log monitoring session on the current application running in the deployed Storm topology by calling "POST /dmon/v1/overlord/storm/logs"; (ii) visualize the list of the available logs that can be obtained from Dmon by calling "GET /dmon/v1/overlord/storm/logs"; (iii) collect the logs of the application that were stored in the specified session and that are available in the platform "GET /dmon/v1/overlord/storm/logs/log_name".

3 USE CASE

The tool has been validated on a case study provided by an industrial partner of the DICE project. The *Trend Topic Detector* topology is part of a wider application for the collection, analysis and visualization of web contents from news and social media. The topology structure includes one spout and 15 bolts. Figure 2 shows the UML Class Diagram defined through the DICE IDE for to run design-time verification with D-VerT. We used TraCT to asses the accuracy of the parameters in the model and validate the results of the verification. Thanks to the tool usage, we were able to refine the model by adjusting most of the values for the *sigma* parameter.

ACKNOWLEDGMENTS

Horizon 2020 project no. 644869 (DICE).

REFERENCES

[1] Marcello M. Bersani, Madalina Erascu, Francesco Marconi, and Matteo Rossi. [n. d.]. D3.5 DICE Verification Tools Initial Version. ([n. d.]). http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/02/D3.5_DICE-verification-tools-Initial-version.pdf

[2] Giuliano Casale, Danilo Ardagna, Matej Artac, Franck Barbier, Elisabetta Di Nitto, Alexis Henry, Gabriel Iuhasz, Christophe Joubert, Jose Merseguer, Victor Ion Munteanu, Juan Perez, Dana Petcu, Matteo Rossi, Chris Sheridan, Ilias Spais, and Daniel Vladušič. 2015. DICE: Quality-Driven Development of Data-Intensive Cloud Applications. In *Proc. of MiSE*. 78–83. www.diceh2020.eu.

[3] Francesco Marconi, Marcello M. Bersani, and Matteo Rossi. 2017. Formal verification of storm topologies through D-VerT. In *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*. 1168–1174.