A Tool for Verification of Big-Data Applications

Jul 21th, 2016

M.M. Bersani, F. Marconi, M.G. Rossi
Politecnico di Milano
Milan, Italy

Madalina Erascu
Institute e-Austria Timisoara &
Western University of Timisoara
Timisoara, Romania

DICE
Horizon 2020 Research & Innovation Action
Grant Agreement no. 644869
http://www.dice-h2020.eu

Funded by the Horizon 2020
Framework Programme of the European Union
Our work at a glance

- Approach and tool for the automated verification of topology-based data-intensive applications.
  - Based (so far) on temporal logic model
  - Performs automated transformation from high level application description to formal model
  - Enables verification of safety properties
Roadmap

- **Context**
  - Quality assurance in DIA

- **Research Design**
  - Research question
  - Our approach

- **Conclusions**
  - Contributions
  - Future works
Quality Analysis and Verification for data-intensive applications

CONTEXT
Formal Verification

- Given a Model M and a Property specification P, verification checks whether P holds in M.
- M and P can be expressed in many different ways
  - various kinds of automata (operational models)
  - various kinds of logics (descriptive models)
Data-Intensive Applications (DIA)
DICE Project

- Horizon 2020 Research & Innovation Action (RIA)
  - Quality-Aware Development for Big Data applications
  - Feb 2015 - Jan 2018, 4M Euros budget
  - 9 partners (Academia & SMEs), 7 EU countries

Imperial College
London

XLAB
NOT IDLE

IEAT
Universidad Zaragoza

POLITECNICO DI MILANO

PRODEVELOP Integración de tecnologías

flexiant™ your cloud simplified
Quality Dimensions in DICE

- Reliability
  - Availability
  - Fault-tolerance

- Efficiency
  - Performance
  - Costs

- Safety & Privacy
  - Verification
  - Data protection
Our positioning in DICE framework (1)
Our positioning in DICE framework (2)

Featuring the DICE H2020 EU Project
Quality Analysis and Verification for data-intensive applications

RESEARCH DESIGN
Research question

“How can we verify safety properties of a data-intensive application?”
State of the art

- Formal verification of distributed systems is a major research area in software engineering.
- Few works trying to address formal verification in the context of DIA:
  - Main focus on verifying *application-independent* properties related to specific frameworks:
    - Reliability and load balancing of MapReduce
    - Validity of messaging flow in MapReduce
  - No modeling and verification of *application-dependent* properties.
- Verification tools have been used as verification engines to build formal verification techniques for UML models:
  - Few of them deal with real-time constraints.
  - Mainly focused on functional requirements.
Our Approach

- Focus on a specific set of technologies
  - Topology-based streaming applications → Apache Storm
- Identify safety issues
- Devise a formal model
  - Having an appropriate level of abstraction
  - Allowing to capture meaningful system behavior and properties
  - Using a formalism that enables automatic verification
- Define a tool-supported mechanism for formal verification
  - Starting from high level application description (annotated UML)
Apache Storm

- Open Source Distributed Stream Processing System
- Analytics, Log Event processing, etc..
- Reliability, at-least-one semantics
- Wide adoption in production
- Main concepts
  - Streams
  - Topologies
Storm Applications

- Applications defined by means of **Topologies**, graphs of computations composed of:
  - **Spouts**
    - Sources of data streams (tuples)
  - **Bolts**
    - Calculate, Filter, Aggregate, Join, Talk to databases
Safety Issues

- Important requirements for streaming applications
  - Latency
  - Throughput

- Critical points
  - Incorrect design of timing constraints
  - Node failures

- Might cause
  - Latency in processing tuples
  - Monotonic growth of the size of used memory (queues).
We want to

- Verify whether a topology reaches an *unwanted configuration*
  - e.g., where bolts are not able to process incoming tuples on time
- Let the user specify the topology by means of high level models (UML)
D-VerT - DICE Verification Tool
D-VerT - DICE-profiled UML Class Diagram
DTSM2Json module

- Relies on Eclipse **UML2** Java library
- “Navigates” DTSM class diagram and extract topology structure and information
- Gathers verification option from Eclipse launch configuration
- Maps topology components to Java objects
- Directly converts Java objects to JSON object via **gson** library
Json2MC - Module

- Python component based on Jinja2 templating engine
- Generates Formal Model based on the content of JSON file and on the selected template (TL or FOL).
Verification Approaches

- **Bounded Satisfiability Checking (BSC)**
  - **Input:**
    - Temporal logic formula (Model)
    - Negated Property over time
  - **Outcome:**
    - SAT $\rightarrow$ counterexample trace
    - UNSAT $\rightarrow$ Property holds for the considered time bound
  - We use Zot verification tool (https://github.com/fm-polimi/zot)

- **Reachability Checking (WIP)**
  - Model defined by FOL Array based system
    - Set of *initial states* and *transitions*
    - Formula defining *undesired states* (*Negated property*)
  - **Outcome:**
    - UNSAFE $\rightarrow$ Trace showing that undesired state are reachable from initial states
    - SAFE $\rightarrow$ No undesired state can be reached from initial states
D-VerT – Output trace

- When at least one queue grows with an unbounded trend
  - an infinite ultimately periodic model is found
  - **Output Parser** provides graphical counterexample trace
CONCLUSIONS
Contributions

- We enabled automatic verification on topology-based streaming applications by
  - Defining a formal model based on temporal logic
  - defining automatic mechanisms for translating to the formal model from a high level description.
  - extending Zot Verification tool to support the formalism and carry out BSC on it
Preliminary results

- Validation through open source and industrial use cases
  - Meaningful qualitative results in identifying critical points in topology design
  - Execution time strongly depends on the size of the topology and on the configurations of single components

<table>
<thead>
<tr>
<th>Topology</th>
<th>Bolts</th>
<th>Time</th>
<th>Max Memory</th>
<th>Outcome</th>
<th>Spurious</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple-DIA-cfg-1</td>
<td>3</td>
<td>60s</td>
<td>104MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>simple-DIA-cfg-2</td>
<td>3</td>
<td>1058s</td>
<td>150MB</td>
<td>UNSAT</td>
<td>N/A</td>
</tr>
<tr>
<td>focused-crawler-complete</td>
<td>8</td>
<td>2664s</td>
<td>448MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-1</td>
<td>4</td>
<td>95s</td>
<td>142MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-2</td>
<td>4</td>
<td>253s</td>
<td>195MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-3</td>
<td>4</td>
<td>327s</td>
<td>215MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-4</td>
<td>4</td>
<td>333s</td>
<td>206MB</td>
<td>SAT</td>
<td>no</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-5</td>
<td>4</td>
<td>3184s</td>
<td>317MB</td>
<td>SAT</td>
<td>yes</td>
</tr>
<tr>
<td>focused-crawler-reduced-cfg-6</td>
<td>4</td>
<td>1060s</td>
<td>229MB</td>
<td>SAT</td>
<td>yes</td>
</tr>
</tbody>
</table>

http://dice-project.github.io/DICE-Verification/
Ongoing and Future works

- Identification and verification of further properties
  - Privacy and Security
- Tool improvements
- Modeling different technologies (Spark, CEP, Tez)
- Developing FOL model
- New theoretical results on the correctness and completeness of the formal analysis
Questions?

Thank you!