Towards the Formal Verification of Data-Intensive Applications Through Metric Temporal Logic

ICFEM 2016, Tokyo
Nov 17\textsuperscript{th}, 2016

Francesco Marconi\textsuperscript{1}, Marcello M. Bersani\textsuperscript{1}, Madalina Erascu\textsuperscript{2} and Matteo Rossi\textsuperscript{1}

\textsuperscript{1}DEIB, Politecnico di Milano, Italy
\textsuperscript{2}Institute e-Austria Timisoara and West University of Timisoara, Timisoara, Romania
Roadmap

- **Context and Motivation**
  - *Data-Intensive Applications*
  - *Streaming DIAs*
  - *Quality issues*

- **Our Approach**
  - *Formal Model*
  - *Decision Procedure*
  - *Implemented tool: D-VerT*

- **Conclusions**
  - *Experimental Analysis*
  - *Future works*
Towards the Formal Verification of Data-Intensive Applications Through Metric Temporal Logic

CONTEXT AND MOTIVATION
DICE Project

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

- Need to process data being
  - Massively large in size
  - Complex
  - Rapidly changing

- Devote most of their processing time to I/O, movement and manipulation of data.

- Rely on so-called "Big data technologies"
The Big Data Landscape

Big Data Landscape 2016 (Version 3.0)

Infrastructure
- Hadoop
- Spark
- NoSQL Databases
- NewSQL Databases
- Graph Databases
- MPP Databases
- Cloud EDW

Analytics
- BI Platforms
- Statistical Computing
- Log Analytics
- Social Analytics
- Machine Learning

Applications
- Sales & Marketing
- Customer Service
- Human Capital
- Legal

Graph Databases
- Neo4j
- OrientBase
- NoTime
- APOC

Security
- TANUM
- Lumigo
- Qatalyst
- DataGravity

Storage
- Cachetron
- Slick
- Ilium
- Semantic DB

App Dev
- Apache
- Cloudera
- Neo4j
- Spark

Crowd-sourcing
- Kaggle
- AlgoRiver
- Analytics
- DataArtisans

Data Sources & APIs
- Apple
- Health
- IOT
- Financial & Economic Data
- Air / Space / Sea
- Location / People / Entities
- Other

Cross-Infrastructure/Analytics
- Google
- Microsoft
- IBM
- SAP
- Hadoop
- Spark
- NoSQL
- NewSQL
- Graph
- MPP
- Cloud EDW

Framework
- PySpark
- Query / Data Flow
- Data Access
- Coordination
- Real-Time

Stat Tools
- Machine Learning
- Data Access
- Query / Data Flow
- Data Sources & APIs

Search
- Hadoop
- Spark
- NoSQL
- NewSQL
- Graph
- MPP
- Cloud EDW

Security
- TANUM
- Lumigo
- Qatalyst
- DataGravity

Open Source
- Google
- Microsoft
- IBM
- SAP
- Hadoop
- Spark
- NoSQL
- NewSQL
- Graph
- MPP
- Cloud EDW

Last Updated 3/23/2016
© Matt Turck (@mattturck), Jim Hao (@jimr Hao), & FirstMark Capital (@firstmarkcap)
Streaming Applications

- Special case of DIAs
- Need to process an (almost) continuous flow of information
  - Stream → unbounded sequence of tuples (messages)
- Usually described by means of a **topology**
  - Graph of computations composed of
    - **input** nodes (source of data streams)
    - **computational** nodes → manipulate data streams
      - Calculate, Filter, Aggregate, Join, Talk to databases, etc
Quality Issues in Streaming DIAs

- Important requirements for streaming applications
  - Latency
  - Throughput

- Critical points
  - incorrect design of timing constraints
  - node failures

- might cause
  - High latency in processing tuples
  - Memory saturation
Questions

- How can we analyze and verify the presence of these kinds of quality (safety?) issues?
  - Which (application dependent) properties could we verify?
  - Associated to which technology?
  - How can we model the system and the properties?
  - How can we automate the verification, providing a “user friendly” support to DIA designers?
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
Towards the Formal Verification of Data-Intensive Applications Through Metric Temporal Logic

PROPOSED SOLUTION
Our Approach

- Focus on a specific set of technologies
  - Topology-based streaming applications

- Identify quality issues

- Select a reference technology → Apache Storm

- Devise a formal model
  - Allowing to capture meaningful system behavior and properties
  - Having an appropriate level of abstraction
  - Using a formalism that enables automatic verification

- Define a tool-supported mechanism for formal verification
  - Starting from high level application description
    - Initial version: JSON format
    - **Current version:** annotated UML Class diagram
Apache Storm

- Open Source Distributed Stream Processing System
- Analytics, Log Event processing, etc..
- Reliability, at-least-one semantics
- Wide adoption in production
- In Storm topologies
  - Source nodes called spouts
  - Computational nodes called bolts
Modeling choices— 1/2

- Allowing for the definition of topologies in a compositional way
  - Formalize behavior of spouts and bolts
  - Use them as building blocks for topologies
- Abstracting away
  - Deployment details
  - Message contents
  - Multi-layered message buffers
Modeling choices – 2/2

- Relevant features modeled for each component
  - evolution of the states
  - timing constraints
  - evolution of its message buffer (input queue)

- Properties to verify
  - “all bolt queues have a bounded occupation level”
Timed counter networks model

- Formal model based on CLTLoc enriched with counters describing:
  - state evolution of components \[ \rightarrow \text{LTL} \subseteq \text{CLTLoc} \checkmark \]
    \[ \left( \prod_{j \in B} \left( \text{process}_j \Rightarrow \left( \text{process}_j \land \text{take}_j \lor (\text{orig} \land \text{process}_j) \right) \land \left( \text{process}_j \lor (\text{emit}_j \lor \neg \text{fail}_j) \right) \right) \right) \]
  - timing constraints \[ \rightarrow \subseteq \text{CLTLoc} \checkmark \]
    \[ \text{process} \land \text{emit} \Rightarrow (t_{\text{phase}} \geq \alpha - \varepsilon) \land (t_{\text{phase}} \leq \alpha + \varepsilon) \]
  - quantities of tuples moving throughout the topology \[ \rightarrow \text{counters!} \]
    \[ \text{add}_j \land \neg \text{take}_j \land \neg \text{startFail}_j \Rightarrow (Xq_j = q_j + r_{\text{add}_j}) \]
    \[ \text{take}_j \Rightarrow (Xq_j = q_j + r_{\text{add}_j} - r_{\text{process}_j}) \]

16
Timed counter networks model
Verifying the property

- We formulated the property check as a **satisfiability** problem
  - Bounded Satisfiability Checking (BSC)

- Goal
  - Find an **ultimately periodic** trace violating boundedness property
    - Having the form \( \alpha(s^\beta)^\omega \)
    - \( \alpha \) \( \rightarrow \) prefix
    - \( s^\beta \) \( \rightarrow \) suffix repeatable infinitely many times (**loop**)

- Rationale
  - If there is a growing trend in the loop \( \rightarrow \) unbounded increase ad infinitum
Decidability issues

- **CLTLoc$^{1,2}$**
  - SAT is decidable and defined over *timed words*
  - Computed through Bounded Satisfiability Checking (BSC)
  - Implemented procedure based on SMT$^3$
    - Using Zot formal verification tool

- **Decidability results cannot** be extended to CLTLoc + counters
  - Contains CLTL over quantifier-free Presburger formulae$^4$

- **We defined a partial assessment method** to guarantee the soundness of the satisfiability outcome.

3. Constraint LTL Satisfiability Checking without Automata, Bersani et al., 2012
4. The effects of bounding syntactic resources on Presburger LTL. Demri, Gascon, 2006
Decision Procedure

- **Given**
  - CLTLoc + counters formula
  - a bound \( k \)

- **Try to build a structure** \( \alpha s \beta s \) with \( |\alpha s \beta s| = k \)
  - If structure is not found (**UNSAT**)
    - No ultimately periodic models of length \( \leq k \) exist
  - If structure is found (**SAT**)
    - Perform the assessment to determine its extensibility to infinite model \( \alpha (s \beta)^\omega \)
      - If check succeeds \( \rightarrow \) outcome is SAT (\( \alpha s \beta \) is counterexample)
      - If check fails \( \rightarrow \) spurious result, must look for another structure
Assessment method

- Provides sufficient condition for extending *ad infinitum* bounded assignment of values to counters
- Intuitively, it checks if in the loop the value of each variable $y$ has the **same shape**
  - It might differ by a non-negative offset $\Delta_y$
D-VerT – DICE Verification Tool
Initial version (April 2016)
D-VerT - DICE Verification Tool

Current Version

https://github.com/dice-project/DICE-Verification/wiki
Experimental results

- Validation through open and closed source 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/
Use case: Focused Crawler Topology

- Typical usage example of Storm
- Fetching and indexing of media items
- From web sources
Use case: Focused Crawler Topology
Output trace
CONCLUSIONS
Wrap up

- Approach for the automated verification of topology-based data-intensive applications.
  - Definition of a formal model (TCN)
    - Extending CLTLoc metric temporal logic with discrete counters
    - Enabling automatic verification of safety properties
  - Definition of a tool-supported mechanism
    - To automatically generate formal models from high level application description and run verification
  - Definition of sufficient conditions for guaranteeing the soundness of the verification results
Future works

- Identification and verification of further properties
- Modeling different technologies
  - Spark, CEP, ...
- New results on the correctness and completeness of the analysis of counter networks
- Tool and model improvements
Thank you
Starting formalism:
Constraint LTL over clocks - CLTLoc

- Extension of LTL with TA clocks, where formulae are
  - Propositions (lightOn, lightOff, buttonOn, buttonOff)
  - Constraints over clocks (c=0, c<1, ...)
  - LTL formulae
    - $X(\phi)$
    - $\phi U \psi$

- CLTLoc$^{1,2}$
  - SAT is decidable and defined over *timed words*
  - Computed through Bounded Satisfiability Checking (BSC)
  - Implemented procedure based on SMT$^{3}$
    - Using Zot formal verification tool

3. Constraint LTL Satisfiability Checking without Automata, Bersani et al., 2012
CLTLoc + counters

- $V$ is a finite set of **variables** over $\mathbb{N}$
- $C$ is a finite set of **clock variables** over $\mathbb{R}$
- $AP$ is a finite set of atomic propositions
- $\theta$ are QFP formulae over terms $\alpha := y \mid Xy$ where $y \in V$

CLTLoc with counters formulae are defined as follows:

$$\phi := p \mid x \sim c \mid \theta \mid \phi \land \phi \mid \neg \phi \mid X\phi \mid Y\phi \mid \phi U \phi \mid \phi S \phi$$

where:
- $p \in AP$, $x \in C$, $c \in \mathbb{N}$, $\sim \in \{<,=\}$
- $X, Y, U, S$ are the usual LTL operators.
Related formalisms

- Timed counter networks are mainly inspired from:
  - Vector Addition Systems with States (VASS)
    - Subclass of counter systems
    - Lossy VASS → take into account number of messages, not their order
    - Only theoretical analysis, do not enable automatic verification
    - Timed counter networks allow to specify timing constraints via clocks
  - Timed Petri Nets
    - Transitions firing with urgent semantics
    - Firing conditions and number of token consumed expressible in a quite rigid way
    - For our model we needed more flexibility
      - Possible occurrence of events
      - Express slightly more elaborate firing conditions