Quantitative Properties: Specification, Verification and Synthesis

Srđan Krstić

advisor: prof. Carlo Ghezzi

co-advisor: Dr. Domenico Bianculli
Quantitative Properties: Specification, Verification and Synthesis

Srđan Krstić

advisor: prof. Carlo Ghezzi

coadvisor: Dr. Domenico Bianculli
Quantitative properties

“Properties that can be objectively expressed using numbers (quantities) with a precisely defined unit of measure.”

– Wikipedia
Quantitative properties

Response Time

Throughput

Availability
Features of Quantitative Properties

- Aggregate transformation
- Timing relation between events
- Numerical bound

“The average response time of a service must not exceed 30 milliseconds, if invoked by a premium customer.”
Features of Quantitative Properties

- Multiplicity of events
  - “At most 3 VM allocations are allowed within 2 minute time window.”
- Time bounded sequence of events
More Generally...

Quantitative Properties

\[ x \propto c \]

...express numerical bound on a certain value

...consider sequence of events

\[ e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n \]

bounded by absolute time \( T \)

\[ e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow p \]

bounded by an event

\[ e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow \ldots \]

or unbounded
More Generally…

Quantitative Properties

\[ \{e_1, e_2, \ldots, e_n\} \quad \text{...compute numerical values from a set of specific events} \]

\[(e_1, e_2, \ldots, e_n) \quad \text{...apply aggregate transformations} \]

\[ \{\text{max()}, \text{avg}(), \text{count}(), \ldots\} \quad \text{...apply aggregate transformations} \]

\[ \text{timing relations between tuples of specific events} \]
Scope

Specification

Verification

Synthesis
Specification
1. Field Study

- Service-Based Applications [1]
- Cloud-Based Systems
- Pervasive Systems

2. Definition and Documentation

semantics 

noun plural but singular or plural in construction
- the study of the meanings of words and phrases in language
- the meanings of words and phrases in a particular context
3. Specification Patterns
4. Decidability and Complexity
5. Usability
Verification
1. Offline Trace Checking
2. Runtime Verification
Synthesis
Inferred specifications: \( S_1, S_2, \ldots, S_p \)

Specification templates:

\[
\begin{aligned}
\phi_{K_1, \ldots, K_m}(p_1, \ldots, p_n) \\
\ldots \\
\phi_{K_1, \ldots, K_m}(p_1, \ldots, p_n)
\end{aligned}
\]
Progress
Specification

Elasticity

Resource Management

Quality of Service
Towards the Formalization of Properties of Cloud-based Elastic Systems

Abstract

Cloud-based elastic systems run on a cloud infrastructure and have the capability of dynamically adjusting the allocation of their resources in response to changes in the workload, in a way that balances the trade-off between the desired elastic behavior of these systems is determined by a combination of factors, including the input workload, the logic of the elastic controller determining the type of resource adjustment, and the underlying technological platform implementing the system. All these factors have to be taken into account to express the desired elastic behavior of a system, as well as to verify whether the system manifests or not such a behavior.

In this paper, we focus on the IaaS layer, and assume, for simplicity and without loss of generality, that resources offered at this level are virtual machines. In particular, we consider elastic systems defined [11] by the (US) National Institute of Standards and Technology (NIST) as:

- Elastic: at any time, the capabilities available to the consumer can remotely access full-fledged software applications; at the Platform-as-a-Service (PaaS) layer, one finds a development environment that provides code in sandboxes and a run-time execution environment; at the Software-as-a-Service (SaaS) layer, one finds a development environment that provides code in sandboxes and a run-time execution environment. To the consumer, the capabilities available can be purchased in any quantity at any time.

I. INTRODUCTION

Cloud computing has become a practical solution to manage and leverage IT resources and services. Cloud platforms offer several benefits, among which the ability to access resources or service applications offered as (remote) services, available on-demand and on-the-fly, and billed according to a pay-per-use model.

Cloud providers offer resources and services at three different layers:

- Infrastructure-as-a-Service (IaaS) layer, one finds a development environment that provides code in sandboxes and a run-time execution environment.
- Platform-as-a-Service (PaaS) layer, one finds a development environment that provides code in sandboxes and a run-time execution environment.
- Software-as-a-Service (SaaS) layer, users can remotely access full-fledged software applications; at the SaaS layer, one finds a development environment that provides code in sandboxes and a run-time execution environment.
Verification

**Specificati**On **Language** f**Or** **Serv**Ice **Compo**Sition **In**Te**r**actions

Metric temporal logic with Aggregates [2]

 Verification

Execution trace

SOLOIST

SMT solver
Verification

QF-EUFIDL

$\Phi$ \rightarrow SMT solver
SMT-based Checking of SOLOIST over Sparse Traces

Marcello M. Bersani¹, Domenico Bianculli², Carlo Ghezzi¹, Srđan Krstić¹, and Pierluigi San Pietro¹

¹ DEEP-SE group - DEIB - Politecnico di Milano, Italy
{bersani,ghezzi,krstic,sanpietr}@elet.polimi.it
² SnT Centre - University of Luxembourg, Luxembourg
domenico.bianculli@uni.lu

Abstract. SMT solvers have been recently applied to bounded model checking and satisfiability checking of metric temporal logic. In this paper we consider SOLOIST, an extension of metric temporal logic with aggregate temporal modalities; it has been defined based on a field study on the use of specification patterns in the context of the provisioning of service-based applications. We apply bounded satisfiability checking to perform trace checking of service execution patterns as specified in SOLOIST. In particular, we focus on sparse traces, i.e., traces in which the number of time instants when events occur is very low with respect to the length of the trace. The main contribution of this paper is an encoding of SOLOIST formulae into formulae of the theory of quantifier-free integer difference logic with uninterpreted function and predicate symbols. This encoding paves the way for efficient checking of SOLOIST formulae over sparse traces using an SMT-based verification toolkit. We report on the evaluation of the proposed encoding, commenting on its scalability and its effectiveness.

1 Introduction

Bounded satisfiability checking [23] (BSC) is a verification technique that complements bounded model checking [9] (BMC): instead of a customary operational model (e.g., a state-transition system) used in BMC, BSC supports the analysis of a descriptive model, denoted by a set of temporal logic formulae. With BSC, verification tasks become suitable instances of the satisfiability problem for quite large formulae (written in a certain logic), which comprehend the model of the system to analyze as well as the requirement(s) to verify. BSC has been successfully applied in the context of metric temporal logics and implemented in ZOT [23], a verification toolset based on SAT- and SMT-solvers, developed within our group.

In this paper we apply BSC to trace checking for the language SOLOIST (Specifi- catiOn Language fOr servIce compOsiOns inTeractions) [8], a metric temporal logic with new, additional temporal modalities that support aggregate operations on events occurring in a given time window. SOLOIST has been defined based on the results of a field study [7] on the use of specification patterns used to express requirements in the context of service-based applications. The study—performed by some of the authors—was aimed at collecting and analyzing a variety of specification patterns used in the context of the provisioning of service-based applications. We apply bounded satisfiability checking to perform trace checking of service execution patterns as specified in SOLOIST. In particular, we focus on sparse traces, i.e., traces in which the number of time instants when events occur is very low with respect to the length of the trace.

The main contribution of this paper is an encoding of SOLOIST formulae into formulae of the theory of quantifier-free integer difference logic with uninterpreted function and predicate symbols. This encoding paves the way for efficient checking of SOLOIST formulae over sparse traces using an SMT-based verification toolkit. We report on the evaluation of the proposed encoding, commenting on its scalability and its effectiveness.
Verification

\[ \neg 28 \]

\[ U_{(0,500)} \]

\[ G_{(0,10000)} \]

\[ p, p, q, r, s \]
Trace checking of Metric Temporal Logic with Aggregating Modalities using MapReduce

Domenico Bianculli¹, Carlo Ghezzi², and Srdan Krstić²

¹ SnT Centre - University of Luxembourg, Luxembourg
domenico.bianculli@uni.lu
² DEEP-SE group - DEIB - Politecnico di Milano, Italy
{ghezzi,krstic}@elet.polimi.it

Abstract. Modern, complex software systems produce a large amount of execution data, often stored in logs. These logs can be analyzed using trace checking techniques to check whether the system complies with its requirements specifications. Often these specifications express quantitative properties of the system, which include timing constraints as well as higher-level constraints on the occurrences of events, expressed using aggregate operators.

In this paper we present an algorithm that exploits the MapReduce programming model to check specifications expressed in a metric temporal logic with aggregating modalities, over large execution traces. The algorithm exploits the structure of the formula to parallelize the evaluation, with a significant gain in time. We report on the evaluation of the implementation—based on the Hadoop framework—of
Future Schedule

T1 Field study - Cloud-based Systems (CBS)

T2 Survey on synthesis

T3 Consider potential new properties

T4 Synthesis of properties of CBS

T5 Adaptation of Offline methods to RV

T6 Field study - Pervasive Systems (PS)
Quantitative Properties: Specification, Verification and Synthesis

Srđan Krstić

advisor: prof. Carlo Ghezzi

co-advisor: Dr. Domenico Bianculli
Acknowledgments

prof. Carlo Ghezzi

Dr. Domenico Bianculli
Additional Slides
## Property Specification Patterns

<table>
<thead>
<tr>
<th>Dwyer et al</th>
<th>Konrad et al</th>
<th>Gruhn et al</th>
<th>Bianculli et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Absence</td>
<td>1 Minimum duration</td>
<td>1 Time bounded existence</td>
<td>1 Avg response time</td>
</tr>
<tr>
<td>2 Universality</td>
<td>2 Maximum duration</td>
<td>2 Time bounded response</td>
<td>2 Counting # events</td>
</tr>
<tr>
<td>3 Existence</td>
<td>3 Bounded recurrence</td>
<td>3 Precedence with delay</td>
<td>3 Avg # events</td>
</tr>
<tr>
<td>4 Bounded existence</td>
<td>4 Bounded response</td>
<td>4 Time restricted precedence</td>
<td>4 Max # events</td>
</tr>
<tr>
<td>5 Precedence</td>
<td>5 Bounded invariance</td>
<td>5 Absolute time</td>
<td>5</td>
</tr>
</tbody>
</table>
Cloud-Based Elastic System

End-Users → Public Interface → Controlled System

- input workload
- invocations

System load → Elastic Controller

- monitoring data
- control action
- maintain QoS
- minimize costs

Public Interface → Cloud Interface

Cloud Interface → add/remove VM

Cloud IaaS Provider → Service Provider

invocations
Property Groups

**Elasticity**
- Eagerness
- Sensitivity
- Plasticity

**Resource Management**
- Precision
- Oscillation
- Resource thrashing
- Cool-down period
- Bounded concurrent adaptations
- Bounded resource usage

**Quality of Service**
- Bounded QoS degradation
- Bounded actuation delay
Examples

Sequence of events

“For a period of 7 days, the application will successfully process a minimum of 500,000 customer orders per day”

Numerical bound

Aggregate transformation
Examples

“The missile avionics system shall update the position of the ailerons exactly 20 times a second.”

Multiplicity of events

Numerical bound
Examples

“The university website shall not have more than 5 hours of scheduled downtime per month and not more than an average of 1 hour of unscheduled downtime per month.”
Related Work
Finkbeiner et al.

- collect statistics over run-time executions
- extend LTL to collect values
- language does not support timing information
Basin et al.

- MFOTL with aggregates
- more aggregates than SOLOIST
- values of the relation parameters vs occurrences
Bauer et al.

- PTLTLFO - past time linear temporal logic with first-order (guarded) quantifiers and counting modality

- lacks timing information and bounded windows
Barre et al.

- MapReduce based approach
- plain LTL (without timing information and aggregates)
- inefficient handling of tuples (no sorting)
- no multi-operand conjunction and disjunction