Online Quantitative Verification: Capabilities and Challenges

Radu Calinescu

Computing Laboratory, University of Oxford
Outline

1 Introduction
   Motivation
   Approach

2 Background
   Quantitative verification with PRISM
   Autonomic computing policies

3 Framework
   Self-* system development
   Policy implementation

4 Challenges

5 Summary
Quantitative verification

Formal technique for establishing quantitative properties of systems that exhibit probabilistic or real-time behaviour

- probability of system being up $\geq 99.9\%$ of the time
- expected length of request queue for a disk drive
Quantitative verification

Formal technique for establishing quantitative properties of systems that exhibit probabilistic or real-time behaviour

- probability of system being up $\geq 99.9\%$ of the time
- expected length of request queue for a disk drive

precise mathematical model of real-world system

formal specification of quantitative system properties

true/false

probability

expected value

exhaustive analysis
Online quantitative verification

Verification of required/desirable quantitative properties is performed at runtime

- analysed model selected based on actual system state
- verification results used to adjust system configuration
Predictable system adaptiveness

(IT) systems required to self-adapt in predictable ways to rapid changes in their workload, environment and objectives

- guaranteed levels of performance and dependability
- compliance with strict constrains

...properties that are traditionally established using (offline) quantitative verification
Predictable system adaptiveness

(IT) systems required to self-adapt in predictable ways to rapid changes in their workload, environment and objectives

- context awareness
- synthesis of reconfiguration “policies” from high-level, multi-objective goals
Approach

Integrate existing quantitative verification tool (PRISM) into the standard autonomic computing architecture
Approach

Integrate existing quantitative verification tool (PRISM) into the standard autonomic computing architecture
The probabilistic model checker PRISM

Developed by the Oxford Quantitative Analysis and Verification Group

Supports multiple types of probabilistic models
- discrete-time Markov chains
- continuous-time Markov chains
- Markov decision processes

plus extensions of these models with costs and rewards

Used to analyse systems from a wide range of application domains
Discrete-/continuous-time Markov chains

\[
\text{DTMC} = (S, s_{\text{init}}, P, L)
\]

- labelling function, \( L : S \rightarrow 2^{AP} \)
- transition probability matrix, \( P : S \times S \rightarrow [0, 1] \)
- initial state, \( s_{\text{init}} \in S \)
- finite set of states
Discrete-/continuous-time Markov chains

\[ \text{DTMC} = (S, s_{\text{init}}, P, L) \]
- labelling function, \( L : S \rightarrow 2^{AP} \)
- transition probability matrix, \( P : S \times S \rightarrow [0, 1] \)
- initial state, \( s_{\text{init}} \in S \)
- finite set of states

\[ \text{CTMC} = (S, s_{\text{init}}, R, L) \]
- transition rate matrix, \( R : S \times S \rightarrow R_+ \)
Example: dynamic power management

![Dynamic power management diagram]

- **Service requester**
- **Power manager (PM)**
- **Service provider (SP)**

- **State information**
- **State-transition commands**

- **Request queue (RQ)**
  - 0 ≤ q ≤ Q_max

- **States**
  - sleep
  - idle
  - busy
  - sp=0
  - sp=1
  - sp=2
Example: dynamic power management

module RQ
    q : [0..Qmax]; // Request queue states

    // State transitions
    [request] true -> 1000/interArrivalTime : (q' = min(q+1,Qmax));
    [serve] q>1 -> (q' = q-1);
endmodule
Example: dynamic power management

```
module SP
  sp : [0..2]; // SP states: 0 – sleep, 1 – idle, 2 – busy

  // State transitions
  [sleep2idle] sp=0 & q=0 -> sleep2idleRate : (sp'=1);
  [sleep2idle] sp=0 & q>0 -> sleep2idleRate : (sp'=2);
  [idle2sleep] sp=1 & q=0 -> idle2sleepRate : (sp'=0);
  [request] sp=1 -> (sp'=2);
  [request] !sp=1 -> true;
  [serve] sp=2 & q>1 -> serviceRate : (sp'=2);
  [serve] sp=2 & q=1 -> serviceRate : (sp'=1);
endmodule
```
Example: dynamic power management

module PM
  p : [0..1]; // PM states: 0 – sleep to idle, 1 – idle to sleep

// State transitions
  [serve] q=1 -> switchToSleepProbability : (p’=1);
  [serve] q=1 -> 1 – switchToSleepProbability : (p’=0);
  [serve] q>1 -> true;
  [request] true -> (p’=0);
  [sleep2idle] q=Qmax -> (p’=p);
  [idle2sleep] p=1 -> (p’=0);
endmodule
Cost/reward extensions

\[ \text{DTMC} = (S, s_{\text{init}}, P, L) \]

- labelling function, \( L : S \rightarrow 2^{AP} \)
- transition probability matrix, \( P : S \times S \rightarrow [0, 1] \)
- initial state, \( s_{\text{init}} \in S \)
- finite set of states

\[ \text{CTMC} = (S, s_{\text{init}}, R, L) \]

- transition rate matrix, \( R : S \times S \rightarrow R^+ \)
- reward structure, \( (\rho, r) \)
- state reward function, \( \rho : S \rightarrow R^+ \)
- transition reward function, \( r : S \times S \rightarrow R^+ \)
Example: power utilisation

```
rewards "power"
  sp=0 : 0.13;        // 0.13W in 'sleep' state
  sp=1 : 0.95;        // 0.95W in 'idle' state
  sp=2 : 2.15;        // 2.15W in 'busy' state
  [sleep2idle] true : 7.0;        // 7J 'sleep' -> 'idle'
  [idle2sleep] true : 0.067;      // 0.067J 'idle' -> 'sleep'
endrewards
```
Quantitative property specification

PCTL—Probabilistic Computational Tree Logic for DTMCs*

CSL—Continuous Stochastic Logic for CTMCs*
Quantitative property specification

PCTL—Probabilistic Computational Tree Logic for DTMCs

CSL—Continuous Stochastic Logic for CTMCs

- PCTL syntax:

  \[ \psi \text{ is true with probability } \sim p \]

  \[ \phi ::= \text{true} \mid a \mid \phi \land \phi \mid \neg \phi \mid P_{\sim p}[\psi] \]  
  (state formulas)

  \[ \psi ::= X \phi \mid \phi U^{\leq k} \phi \mid \phi U \phi \]  
  (path formulas)

- where a is an atomic proposition, used to identify states of interest, \( p \in [0,1] \) is a probability, \( \sim \in \{<,>,\leq,\geq\} \), \( k \in \mathbb{N} \)
Quantitative property specification

PCTL—Probabilistic Computational Tree Logic for DTMCs*
CSL—Continuous Stochastic Logic for CTMCs*

* augmented with costs/rewards

\[ \phi ::= \ldots \mid P_{\sim p}[\psi] \mid R_{\sim r}[l=k] \mid R_{\sim r}[C\leq k] \mid R_{\sim r}[F\phi] \]

- "instantaneous"
- "cumulative"
- "reachability"

where \( r \in \mathbb{R}_{\geq 0} \), \( \sim \in \{<,>,\leq,\geq\} \), \( k \in \mathbb{N} \)
Example: power use; request queue length
The “knowledge” module

Knowledge = (S, C, f)

- operational model, \( f : S \times C \rightarrow S \)
- configuration (modifiable system parameters)
- state (“read-only” system parameters)
Utility-function autonomic policies

Knowledge=$(S, C, f)$

- operational model, $f : S \times C \rightarrow S$
- configuration (modifiable system parameters)
- state (“read-only” system parameters)

Given a utility function

$$utility : S \times C \rightarrow R_+,$$

adjust the configurable system parameters such as to maximise the system utility “at all times”

$$\left[ \text{for } s_0 \in S, \text{ find } c \in C \text{ s.t. } c = \arg\max_{x \in C} utility(f(s_0, x), x) \right]$$
Example: multi-objective utility function

\[ utility = \sum_{i=1}^{n} w_i \text{objective}_i \]
Self-* system development

Key
automated step
computer-assisted step
manual step

Generation
- Markov chain
- Knowledge module
  - model-driven generation
- Legacy component adaptors

Deployment
- Configured autonomic manager
  - autonomic manager configuration
- Manageable components
  - adaptor deployment

Exploitation
- Self-* system
  - policy specification
  - component discovery

Challenges
- Generation
- Deployment
- Exploitation

Framework
- Self-* system development
- Policy implementation

Background
- Quantitative verification with PRISM
- Autonomic computing policies

Introduction
- Motivation
- Approach

Summary
Self-* system development

PRISM discrete-/continuous-time Markov chain
- available from the formal verification of the system
- newly developed
Automated transformation, except for the partition of the Markov chain parameters into state and configuration
Self-* system development

Off-the-shelf tools (XSLT engine, data type generator) used to generate most adaptor code
Self-* system development

Knowledge module supplied at runtime to autonomic manager instance
Self-* system development

Adaptor deployment leads to automatic component discovery by the autonomic manager
Utility-function policy specified by system administrator - multi-objective utility function defined in terms of cost/reward structures from the PRISM Markov chain
Policy implementation

Periodically and/or when the autonomic manager is notified about system changes:

1. **foreach** component \( c \) in the policy scope **do**
2. extract parameterised model of \( c \) from the knowledge module
3. get state parameters of \( c \) from the manageability adaptors
4. evaluate quantitative properties used in the utility function
5. choose configuration parameters that maximise the utility of \( c \)
Example: dynamic power management

- Power-driven objective
- Response-driven objective

Combined utility ($w_1 = w_2 = 100; l_1 = 9; u_1 = 11; l_2 = 0; u_2 = 1200$)
Example 2: cluster availability management

Multi-objective utility function:

1. achieve target availability in the presence of failures and variations in the number of requested servers
2. minimise number of allocated servers
Challenges: inherited from offline verification

State-space explosion
- new model checking techniques still needed

Expert knowledge required to produce "good" models
- more models should be built as part of the system development process
Challenges: specific to online verification

Model checkers not typically intended for online use
  • use command-line interfaces (lower-level APIs better)

Prohibitive analysis time
  • pre-compute/cache analysis results; hybrid approaches

Local optima (unless all possible configurations verified)
  • offline assessment to ensure solution is effective

Utility-function definition
  • close to natural language property/utility specification?
Summary

Increasing need for IT systems to adapt in predictable, dependable ways to changes in their state, objectives and environment

Quantitative verification reached a level of maturity that enables its online use to achieve such adaptiveness in certain scenarios

Interesting research work required to address challenges posed by online quantitative verification
Thank you

Questions?

Further reading
