Online Quantitative Verification: Capabilities and Challenges

Radu Calinescu

Computing Laboratory, University of Oxford

Outline

Quantitative verification

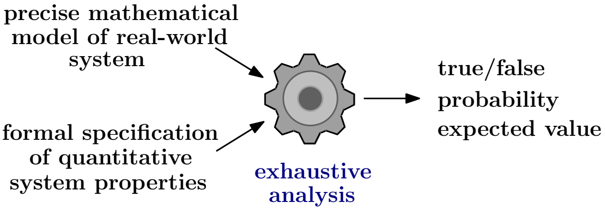
Formal technique for establishing quantitative properties of systems that exhibit probabilistic or real-time behaviour

- 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

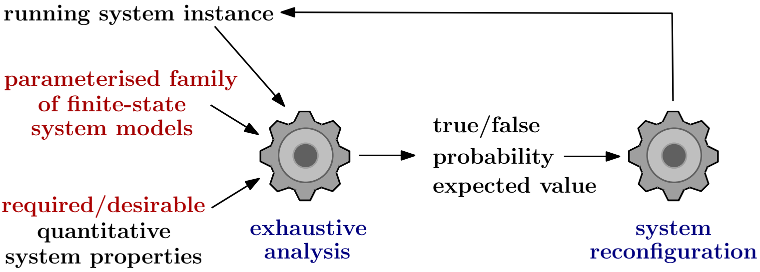
- probability of system being up $\geq$ 99.9% of the time
- expected length of request queue for a disk drive



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



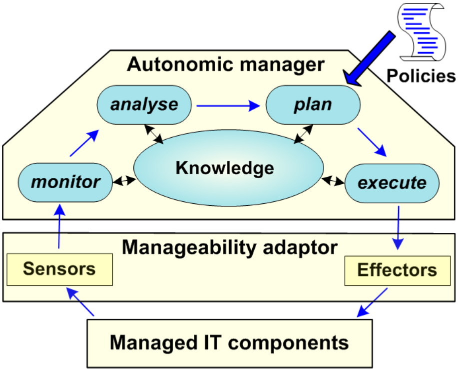
Predictable system adaptiveness

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

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

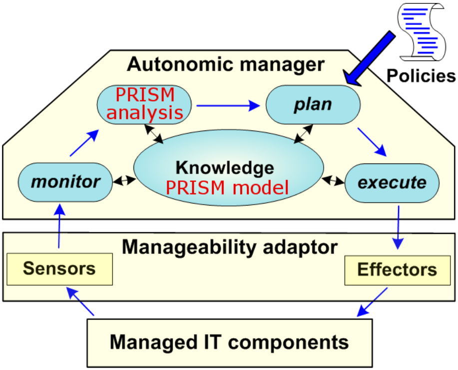
Approach

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

$$\mathrm{DTMC} = (S, s_{\mathrm{init}}, \mathbf{P}, L)$$

- labelling function, $L : S \to 2^{AP}$
- transition probability matrix, $\mathbf{P} : S \times S \to [0, 1]$
- initial state, $s_{\mathrm{init}} \in S$
- finite set of states

Discrete-/continuous-time Markov chains

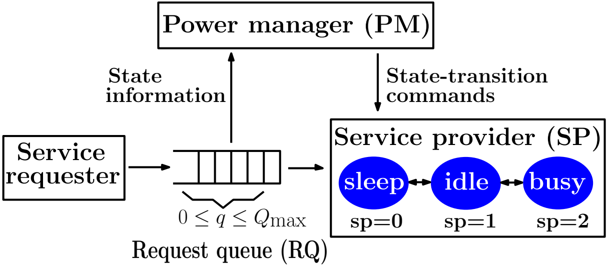
$$\mathrm{DTMC} = (S, s_{\mathrm{init}}, \mathbf{P}, L)$$

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

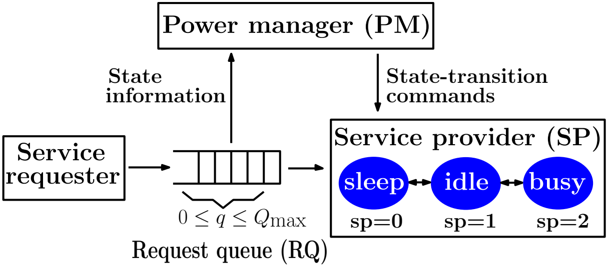
$$\mathrm{CTMC} = (S, s_{\mathrm{init}}, \mathbf{R}, L)$$

- $\mathbf{R}$ — transition rate matrix, $\mathbf{R} : S \times S \rightarrow R_{+}$

Example: dynamic power management

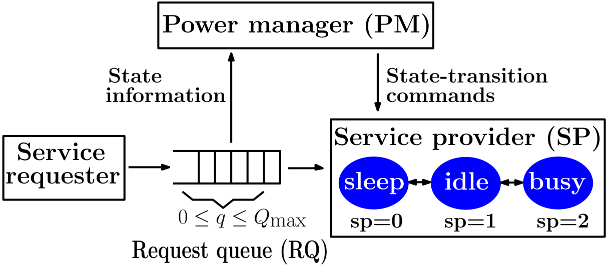


Example: dynamic power management




```
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
```

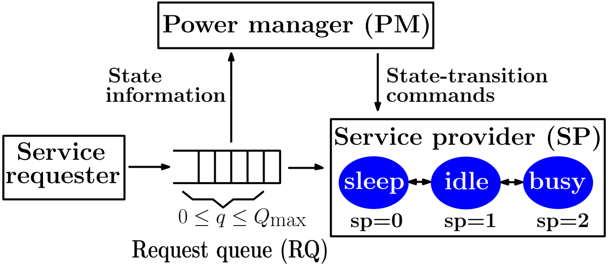
Example: dynamic power management




```
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

$$\text{DTMC} = (S, s_{\text{init}}, \mathbf{P}, L)$$

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

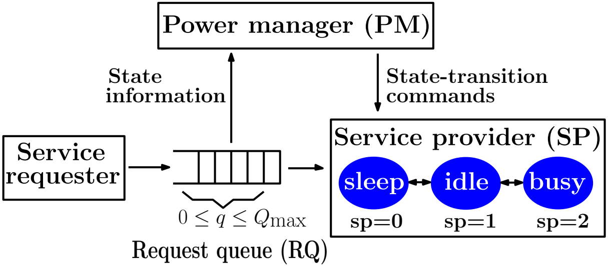
$$\text{CTMC} = (S, s_{\text{init}}, \mathbf{R}, L)$$

- transition rate matrix, $\mathbf{R} : S \times S \rightarrow R_+$

$$\text{reward structure} = (\underline{\rho}, \mathbf{r})$$

- transition reward function, $r : S \times S \rightarrow R_+$
- state reward function, $\underline{\rho} : S \rightarrow R_+$

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
```

R. Calinescu
(University of Oxford)

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:
  - ϕ ::= true | a | $\phi \wedge \phi$ | $\neg\phi$ | $P_{\sim p}$ [ψ] (state formulas)
  - ψ ::= X ϕ | $\phi\ U^{\leq k}\ \phi$ | $\phi\ U\ \phi$ (path formulas)
  - where a is an atomic proposition, used to identify states of interest, p $\in$ [0,1] is a probability, $\sim\ \in \{<,>,\leq,\geq\}$, $k \in \mathbb{N}$

ψ is true with probability ~p

"next"

"bounded until"

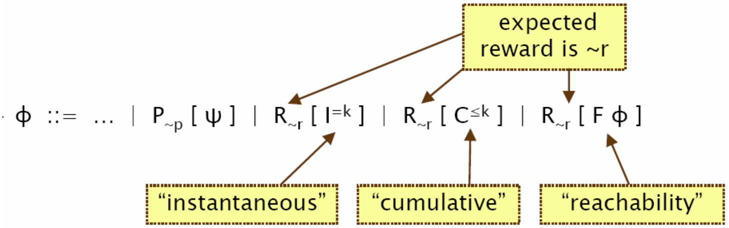
"until"

Quantitative property specification

PCTL—Probabilistic Computational Tree Logic for DTMCs*

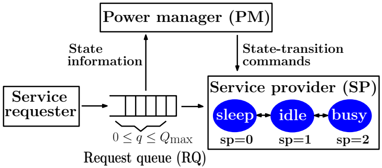
CSL—Continuous Stochastic Logic for CTMCs*

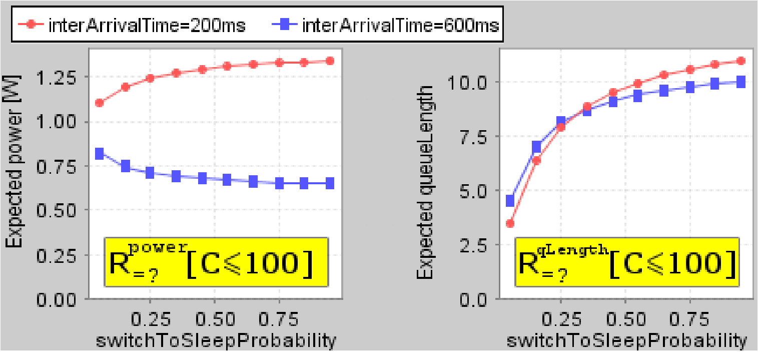
* augmented with costs/rewards



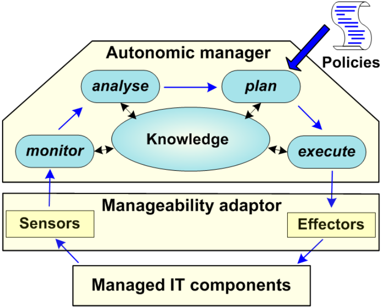
where $r \in \mathbb{R}_{\geq 0}$, $\sim \in \{<, >, \leq, \geq\}$, $k \in \mathbb{N}$

Example: power use; request queue length





The “knowledge” module



$$\text{Knowledge} = (S, C, f)$$

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

Utility-function autonomic policies

Knowledge=(S, C, f)

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

Given a utility function

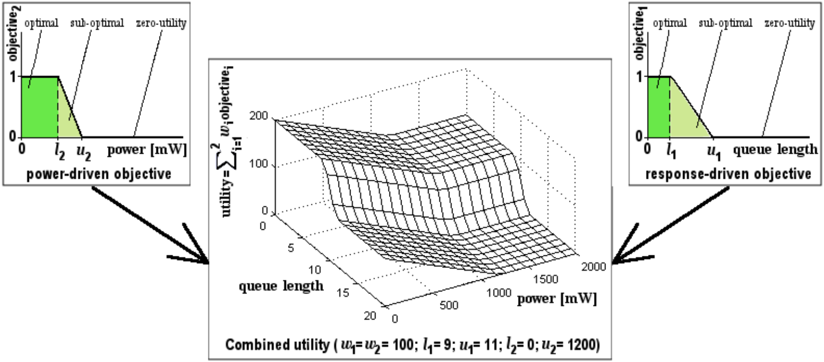
$$utility : S \times C \rightarrow R_+,$$

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

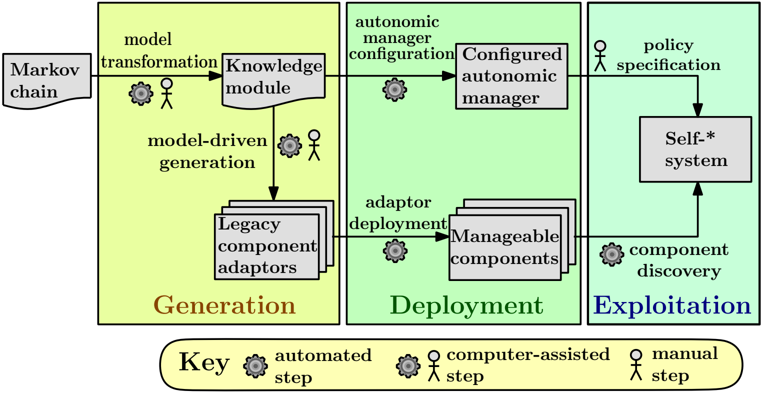
$$\left[\text{for } \mathbf{s}_0 \in S, \text{ find } \mathbf{c} \in C \text{ s.t. } \mathbf{c} = \underset{\mathbf{x} \in C}{\mathbf{argmax}}\; utility(f(\mathbf{s}_0, \mathbf{x}), \mathbf{x})\right]$$

Example: multi-objective utility function

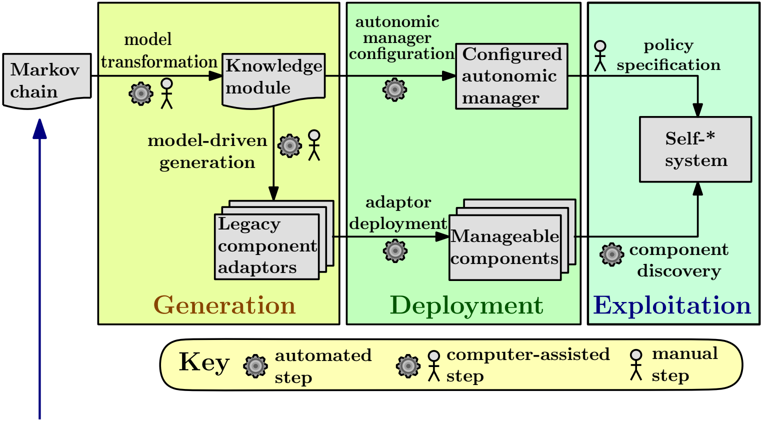
$$utility = \sum^{n} w_i objective_i$$



Self-* system development

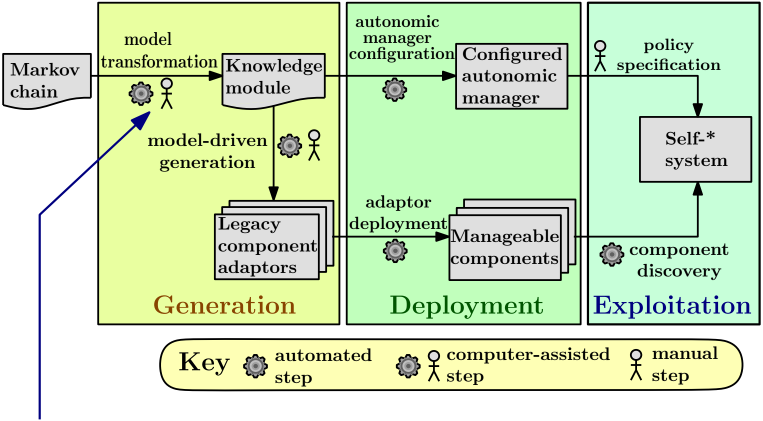


Self-* system development



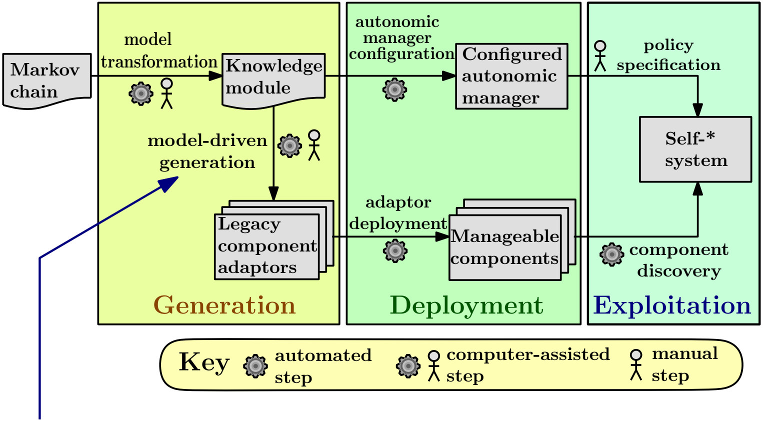
PRISM discrete-/continuous-time Markov chain
- available from the formal verification of the system
- newly developed

Self-* system development



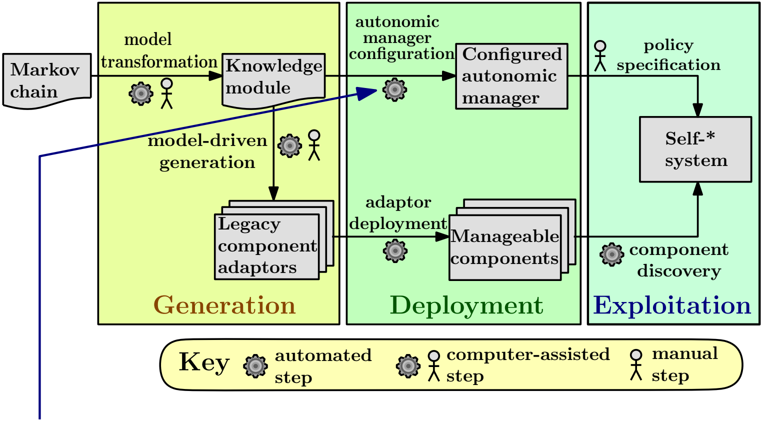
Automated transformation, except for the partition of the Markov chain parameters into state and configuration

Self-* system development



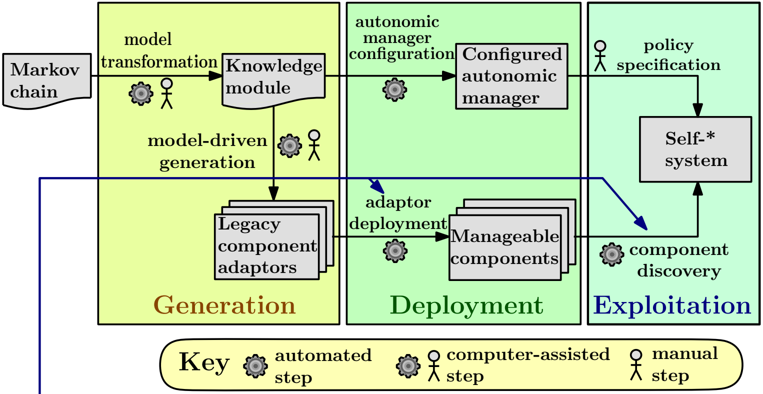
Off-the-shelf tools (XSLT engine, data type generator) used to generate most adaptor code

Self-* system development



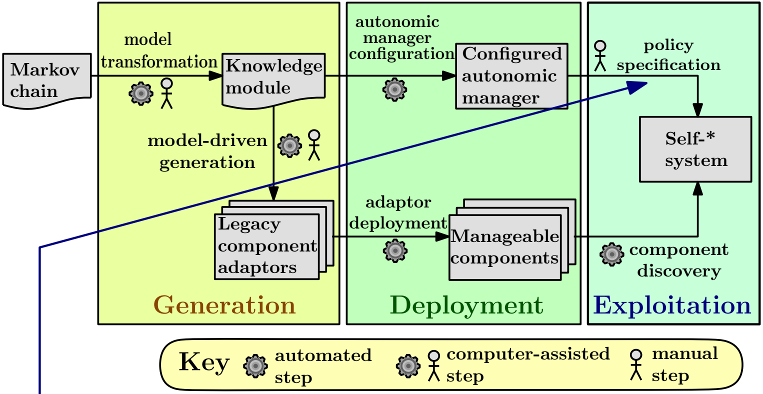
Knowledge module supplied at runtime to autonomic manager instance

Self-* system development



Adaptor deployment leads to automatic component discovery by the autonomic manager

Self-* system development



Utility-function policy specified by system administrator
- multi-objective utility function defined in terms of cost/reward structures from the PRISM Markov chain

utility function example

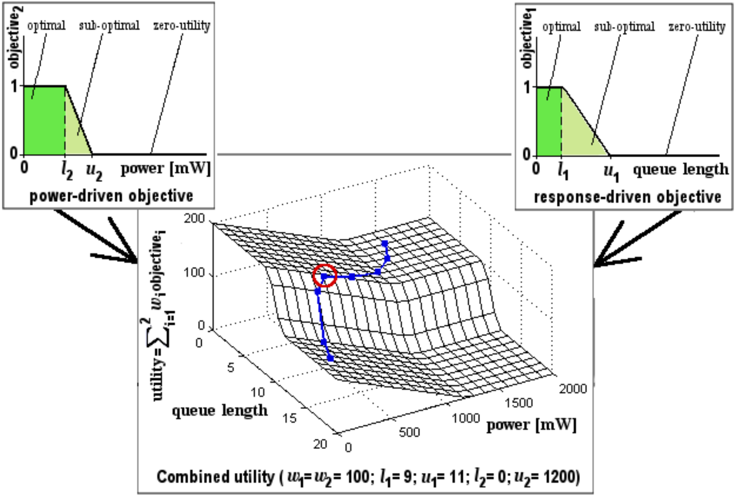
Policy implementation

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

foreach component *c* in the policy scope **do**
    extract parameterised model of *c* from the knowledge module
    get state parameters of *c* from the manageability adaptors
    evaluate quantitative properties used in the utility function
    choose configuration parameters that maximise the utility of *c*

R. Calinescu
(University of Oxford)

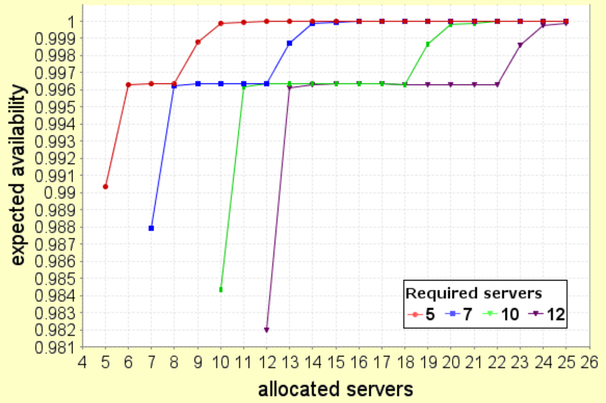
Example: dynamic power management



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

R. Calinescu – General-Purpose Autonomic Computing, In: M. Denko et al., *Autonomic Computing and Networking*, Springer, April 2009, pp. 3–29.

R. Calinescu and M. Kwiatkowska – Using Quantitative Analysis to Implement Autonomic IT Systems, *Proc. 31st Intl. Conf. Software Eng.* (ICSE 2009).