Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration

Article · June 2014

DOI: 10.1145/2593929.2593932

CITATIONS
32

READS
194

3 authors, including:

Simos Gerasimou
Cyprus University of Technology
35 PUBLICATIONS **254** CITATIONS

SEE PROFILE

Radu Calinescu
The University of York
143 PUBLICATIONS **1,775** CITATIONS

SEE PROFILE

Efficient Runtime Quantitative Verification Using Caching, Lookahead and Nearly-Optimal Reconfiguration

Simos Gerasimou
Department of Computer Science
University of York, UK

Radu Calinescu
Department of Computer Science
University of York, UK

Alec Banks
Defence Science and Technology Laboratory
Ministry of Defence, UK

ABSTRACT

Self-adaptive systems used in safety-critical and business-critical applications must continue to comply with strict non-functional requirements while evolving in order to adapt to changing workloads, environments, and goals. Runtime quantitative verification (RQV) has been proposed as an effective means of enhancing self-adaptive systems with this capability. However, RQV frequently fails to provide the fast response times and low computation overheads required by real-world self-adaptive systems. In this paper, we investigate how three techniques, namely caching, lookahead and nearly-optimal reconfiguration, and combinations thereof, can help address this limitation. Extensive experiments in a case study involving the RQV-driven self-adaptation of an unmanned underwater vehicle indicate that these techniques can lead to significant reductions in RQV response times and computation overheads.

1. INTRODUCTION

Self-adaptive software and software-controlled systems are increasingly used in applications that are safety or business critical [4]. The fact that such systems evolve continually in response to changes in environment or user goals raises significant concerns about their ability to maintain compliance with essential reliability, performance and other quality-of-service (QoS) requirements. In recent work, we co-proposed *runtime quantitative verification* (RQV) as a means of addressing this concern [5]. RQV operates by continually verifying stochastic (e.g., Markovian) models of the system, to identify or predict departures from the required QoS properties of the system, and thus to drive self-adaptation towards restoring or preserving QoS requirement compliance.

The successful application of RQV in domains ranging from dynamic power management [11] and service-based systems [6, 12] to cloud infrastructure management [17] show the effectiveness of the approach in guaranteeing QoS requirement compliance for critical self-adaptive systems. This is particularly true when the stochastic models being verified at runtime are updated continually based on observations of the system and its components [7, 8, 12].

Notwithstanding its strenghts, RQV is affected by *state explosion*, a problem common to all model checking techniques. This exponential increase in the number of model states with the size of the self-adaptive system limits the applicability of RQV to small-sized systems and simple scenarios [5]. Addressing this limitation represents a significant research challenge. To tackle this challenge, the RQV community has recently proposed variants of the approach that are: (i) *compositional*, by using assume-guarantee model checking to verify component-based systems one component at a time [9, 23]; (ii) *incremental*, by deriving the current verification results from those obtained in previous verification steps [17, 24]; or (iii) *pre-computation*-based, by deriving algebraic representations of QoS properties for fast runtime evaluation [13]. Each of these techniques reduces the time and/or resources needed to perform an *RQV step* (i.e., the RQV operations triggered by a change in the system) for certain self-adaptation scenarios and for specific types of stochastic models and properties.

In this paper we introduce a set of complementary techniques aimed at improving RQV efficiency further, including through integration with the RQV variants mentioned above. These techniques are adapted from other areas of software engineering and, to the best of our knowledge, have not been applied to RQV previously.

First, we consider the *caching* of recent verification results. Since changes in real-world systems are often (though by no means always) localised, there is a possibility that verification results from recent RQV steps could be reused if retained for some time. Similar to other applications of caching, the aim is to reduce RQV *response time* (i.e., the time required to perform an RQV step) and CPU usage at the expense of using additional memory.

Second, we augment RQV with *limited lookahead*, which involves using spare CPU cycles to continuously pre-verify stochastic models deemed likely to arise in the future. Since some RQV steps may require the verification of models that were already pre-verified, the technique has the potential to reduce RQV response time at the expense of increased use of CPU and memory.

Finally, we combine RQV with *nearly-optimal reconfiguration*, a technique that terminates RQV steps as soon as they identify a system configuration that (a) satisfies QoS requirements, and (b) has a similar cost to the best cost seen over a pre-defined time interval.

SEAMS '14 Hyderabad, India

The main contributions of the paper are:

- The integration of RQV with caching, limited lookahead and nearly-optimal reconfiguration.
- The extension of the open-source platform MOOS-IvP (`http://oceanai.mit.edu/moos-ivp`) for the development of autonomous systems with runtime model checking capabilities.
- The first application of the RQV to the unmanned underwater vehicle (UUV) domain.
- A realistic case study from the UUV domain, used to carry out a preliminary assessment of the effectiveness of RQV extended with caching, limited lookahead, nearly-optimal reconfiguration, and combinations thereof.

The rest of the paper is structured as follows. Sections 2 and 3 introduce the self-adaptive UUV system used in our case study, and the theoretical background for the other parts of the paper, respectively. The new RQV techniques are presented in Section 4, followed by a description of their implementation within a widely used development environment from the UUV domain in Section 5. Our experimental results are analysed in Section 6, and Sections 7 and 8 conclude the paper with a discussion of related work, and with a brief summary and an overview of our planned future work, respectively.

2. SELF-ADAPTIVE UUV SYSTEM

We will use a self-adaptive UUV system to evaluate the RQV techniques proposed in the paper. UUVs are increasingly used in a wide range of oceanographic and military tasks, including oceanic surveillance (e.g., to monitor pollution levels and ecosystems), undersea mapping, and mine detection. Limitations intrinsic to the environment in which these vehicles operate (e.g., impossibility to maintain UUV-operator communication during missions and high frequency of unexpected changes) require that UUV systems are self-adaptive [27]. These systems are also safety critical (e.g., when used for mine detection and surveillance of ecosystems that should not be impacted) and/or business critical, since UUVs are often expensive equipment that should not be lost during missions.

The self-adaptive system in our study consists of a UUV used to carry out a surveillance and data gathering mission. The UUV is equipped with $n \geq 1$ on-board sensors that can measure the same attribute of the ocean environment (e.g., water current, salinity or thermocline). When used, the sensors take measurements with different, variable rates $r_1, r_2, \ldots, r_n$, consume different amounts of energy $e_1, e_2, \ldots, e_n$ for each measurement. These measurements have an accuracy that depends on the UUV speed s, and the (speed-dependent) probabilities $p_1, p_2, \ldots, p_n$ that the sensors produce measurements that are sufficiently accurate for the purpose of mission can be calculated from the technical specifications of the sensors. Finally, the n sensors can be switched on and off individually (e.g., to save battery power when not required), but these operations consume an amount of energy given by $e_1^{\text{on}}, e_2^{\text{on}}, \ldots, e_n^{\text{on}}$ and $e_1^{\text{off}}, e_2^{\text{off}}, \ldots, e_n^{\text{off}}$, respectively.

The UUV is required to self-adapt to changes in the observed sensor measurement rates r_i, $1 \leq i \leq n$, and to sensor failures by dynamically adjusting

(a) the UUV speed s

(b) the sensor configuration $x_1, x_2, \ldots, x_n$ (where $x_i = 1$ if the i-th sensor is on and $x_i = 0$ otherwise)

so that the UUV complies with the following requirements at all times:

R1: The UUV should take at least 20 measurements of sufficient accuracy for every 10 metres of mission distance.

R2: The energy consumption of the sensors should not exceed 120 Joules per 10 surveyed metres.

R3: If requirements R1 and R2 are satisfied by multiple configurations, the UUV should use one of these configurations that minimises the cost function

$$cost = w_1 E + w_2 s^{-1}, \tag{1}$$

where E represents the energy consumed by the sensors to survey a 10m mission distance, and $w_1, w_2 > 0$ represent weights that reflect the relative importance of carrying out the mission with reduced battery usage and completing the mission faster.

3. BACKGROUND

3.1 Continuous-Time Markov Chains

Definition 1. A continuous-time Markov chain (CTMC) over a set of propositions AP is a tuple

$$M = (S, s_0, \mathbf{R}, L), \tag{2}$$

where:

- S is a finite set of states;
- $s_0 \in S$ is the initial state;
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is a transition rate matrix;
- $L : S \to 2^{AP}$ is a labelling function which assigns a set of atomic propositions from AP to each state in S.

For any state $s_1 \in S$, the probability that the CTMC will transition from state s_1 to another state within $t > 0$ time units is $1 - e^{-t \cdot \sum_{s \in S} \mathbf{R}(s_1, s)}$, and the probability that the new state is $s_2 \in S$ is given by $\mathbf{R}(s_1, s_2) / \sum_{s \in S} \mathbf{R}(s_1, s)$.

Quantitative or *probabilistic* model checkers (e.g., PRISM [22] and MRMC [18]) operate on Markovian models expressed in a high-level, state-based language. Given a CTMC description in this language, the low-level representation (2) is derived automatically. Our work uses the probabilistic model checker PRISM [22], which supports the analysis of CTMCs augmented with cost/reward structures, as described below.

Definition 2. A cost/reward structure over a continous-time Markov chain $M = (S, s_0, \mathbf{R}, L)$ is a pair of real-valued functions $(\underline{\rho}, \iota)$, where

- $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$ is a *state reward function* that defines the rate at which the reward is obtained while the CTMC is in state s;
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a *transition reward function* that defines the reward obtained each time a transition occurs.

Example 1. Fig. 1 depicts the CTMC model M_i of the i-th sensor of the UUV from our case study. The CTMC starts in state 0 and transitions in state 1 or state 6 if the sensor is switched on ($x_i = 1$) or switched off ($x_i = 0$), respectively. The transition between states 1 and 2 corresponds to the sensor performing a measurement with rate r_i. "Sufficiently

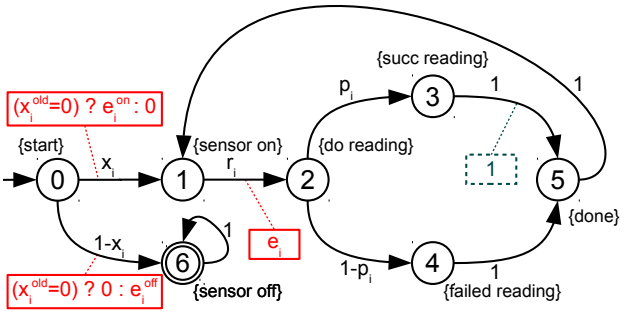

Figure 1: CTMC model M_i of the i-th UUV sensor

accurate" measurements (in the sense defined in Section 2) are modelled by the transition between states 2 and 3, which is taken with probability p_i, whereas the transition between states 2 and 4 corresponds to inaccurate measurements. A sensor that is active carries on performing measurements (at least until the next RQV step), as modelled by the transition between states 5 and 1. The CTMC model is augmented with two cost/reward structures, whose non-zero elements are shown in Fig. 1 in rectangular boxes, and dashed rectangular boxes, respectively. The former, "*energy*" structure associates the energy used to switch the sensor on (i.e., e_i^{on}) and off (i.e., e_i^{off}) and to perform a measurement (i.e., e_i) with the CTMC transitions that model these events. Note that the expression "$condition?a : b$" was used to indicate that the energy e_i^{on} is consumed only if the previous state of the sensor was "off", i.e., if $x_i^{\text{old}} = 0$, whereas the energy e_i^{off} is used only if the opposite is true ($x_i^{\text{old}} = 1$). As concerns the latter, "*measurement*" cost/reward structure, it associates a reward of 1 with the transition that corresponds to an accurate measurement.

3.2 Continuous stochastic logic

Cost/reward-augmented versions of continuous stochastic logic (CSL) [1] are used to specify the quantitative properties to analyse for CTMC models. In our work, we use the cost-reward augmented CSL variant with the syntax from [20], as detailed below.

Definition 3. Let AP be a set of atomic propositions, $a \in AP$, $p \in [0,1]$, I an interval in $\mathbb{R}$ and $\bowtie \in \{\geq, >, <, \leq\}$. Then a state-formula Φ and a path formula ϕ in CSL are defined by the following grammar:

$$\begin{aligned} \Phi &::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\bowtie p}\phi \mid S_{\bowtie p}\Phi \\ \phi &::= X\Phi \mid \Phi \cup^{\leq I} \Phi \end{aligned} \tag{3}$$

and the cost/reward augmented CSL state formulae are defined by the grammar:

$$\Phi ::= R_{\bowtie r}[I^{=T}] \mid R_{\bowtie r}[C^{\leq T}] \mid R_{\bowtie r}[F\Phi] \mid R[S] \tag{4}$$

CSL formulae are interpreted over states of a CTMC model. Path formulae only occur inside the probabilistic operator P. This operator defines upper or lower bounds on the probability of system evolution. For instance, a state s satisfies a formula $P_{\bowtie p}[\phi]$ if the probability of the future evolution of the system meets the bound $\bowtie p$. For a path, the "next" formula $X\Phi$ holds if Φ is satisfied in the next state; the "bounded until" formula $\Phi_1 \cup^{\leq I} \Phi_2$ holds if before $\Phi 2$ becomes true at time step x, where $x \leq I$, Φ_1 is satisfied

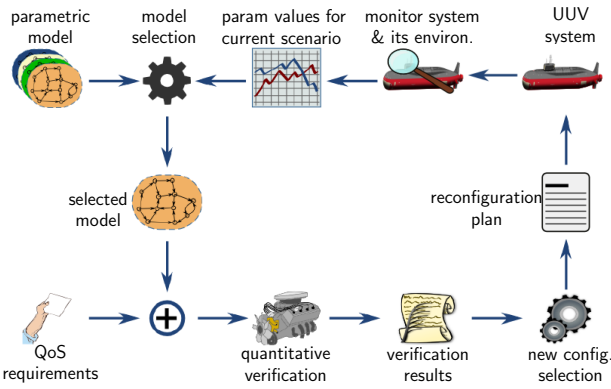

Figure 2: RQV-based self-adaptive system

continuously in the interval $[0, x)$. Note that if $I = \infty$, the formula is termed "unbounded until".

The high-level interpretation of the cost/reward operator R has as follows: $R_{\bowtie r}[I^{=T}]$ denotes the expected value of the reward at time instant T; $R_{\bowtie r}[C^{\leq T}]$ denotes the expected cumulative reward up to time T; $R_{\bowtie r}[F\Phi]$ indicates the expected cumulative reward before reaching a state that satisfies Φ; and $R[S]$ indicates the average expected reward in the long-run. For a formal analysis of the CSL semantics, see [21].

Example 2. Consider again the UUV system from our case study, and the CTMC model M_i of its i-th sensor from Fig. 1. The model M of the entire n-sensor system is obtained through the parallel composition of the n sensor models, i.e., $M = M_1 \parallel M_2 \parallel \ldots \parallel M_n$, and the requirements R1 and R2 can be expressed formally in CSL as

R1: $R_{\geq 20}^{measurement}[C^{\leq 10/s}]$

R2: $R_{\leq 120}^{energy}[C^{\leq 10/s}]$

where $10/s$ represents the time taken by the UUV to cover 10m at speed s. As concerns requirement R3, the expected energy consumption for the cost function (1) is produced by the evaluation of the CSL formula

$$R_{=?}^{energy}[C^{\leq 10/s}]$$

over the model M.

3.3 Runtime Quantitative Verification

Runtime quantitative verification (RQV) [5] is an approach to implementing the closed control loop of self-adaptive systems in which the adaptation decisions are driven by the continual verification of stochastic models. RQV was introduced in [11, 12] and further refined in [6, 13, 17].

As shown in Fig. 2, the approach involves monitoring the system (e.g., the UUV) and its environment continuously, to identify relevant changes and quantify them using fast on-line learning techniques. When these changes are significant and/or periodically, an RQV step is triggered, to identify which scenario the system operates in, and to select the associated model from a family of system models (i.e., a *parametric model*) that correspond to different such scenarios. The selected model is then analysed using quantitative verification, to identify and/or predict violations of QoS requirements such as response time, availability and cost. When QoS violations are identified or predicted, the

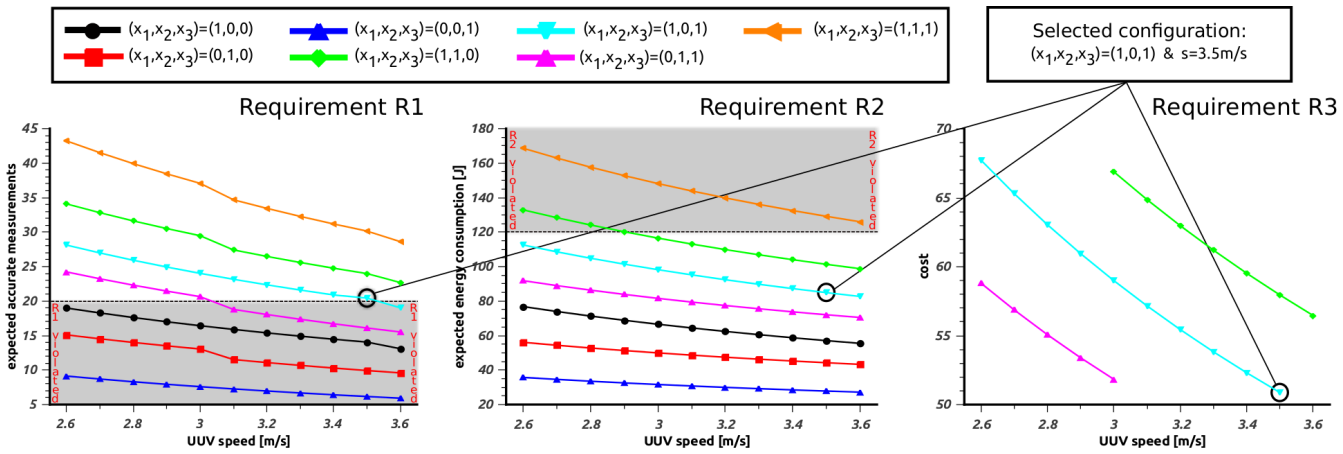

Figure 3: RQV verification step for the UUV system, with shaded areas used to indicate configurations that violate requirements R1 and R2

Table 1: Parameter values for the UUV system analysed in Fig. 3

Sensor	e_i [J]	e_i^{old} [J]	e_i^{new} [J]	r_i [s^{-1}]	x_i^{old}
1	3	15	2	5	1
2	2	10	1.5	4	0
3	1	5	1	4	0

verification results support the selection of a new set of values for the configurable system parameters, so that enforcing this new configuration is *guaranteed* to restore or maintain compliance with QoS requirements, respectively.

Example 3. Fig. 3 illustrates the analysis carried out by an RQV step for an instance of the UUV system from our case study that has $n = 3$ sensors with the parameters from Table 1.

4. TECHNIQUES FOR EFFICIENT RQV

4.1 Caching

Our use of caching to reduce the response time and the computational overhead of RQV employs a standard cache for the storage of $\langle(modelParams, propIdx), (res, timestamp)\rangle$ entries from recent RQV steps, where:

- *modelParams* represents the set of all parameter values required to select the relevant instance of the parametric model of the system—for our case study, this set contains the values of x_i^{old}, x_i, r_i and p_i from Fig. 1, $1 \leq i \leq n$ (since e_i, e_i^{on} and e_i^{off} are constant values);
- *propIdx* is the index of the verified CSL property, e.g., i for the CSL formula associated with i-th requirement;
- *res* represents the result obtained from the quantitative verification of the CSL property with index *propIdx* against the model with parameters *modelParams*;
- *timestamp* is the latest time when this entry was used.

Algorithm 1 shows the pseudocode for this cache-enabled version of RQV. Starting with an empty cache (line 1), the technique uses a VERIFY function that stores new verification results obtained in line 9 into the cache (line 13) for future use (lines 5–7).

Algorithm 1 RQV with LRU-replacement caching

```
1:  LRUCache ← {}
2:  function VERIFY(modelParams, propIdx)
3:      key ← (modelParams, propIdx)
4:      if LRUCache.containsKey(key) then
5:          entry ← LRUCache.get(key)
6:          entry.timestamp ← NOW
7:          result ← entry.res
8:      else
9:          result ← QV(modelParams, propIdx)
10:         if LRUCache is full then
11:             Evict LRUCache entry using the LRU policy
12:         end if
13:         LRUCache.add(key, (result, NOW))
14:     end if
15:     return result
16: end function
```

4.2 Lookahead

Our use of lookahead involves taking advantage of spare computational resources to precompute the quantitative verification results associated with *system and environment states* "close" to the current state. Under the assumption that real-world systems are often (though not always) evolving through small changes, this will ensure that verification results required during an RQV step will sometimes be already available. The effect is a decrease in the time required to take self-adaptation decisions, at the expense of using additional computational resources. In our case study, for example, this may take advantage of periods of time when the CPU is idle. In other applications, the pre-computations of potentially useful quantitative verification results may be delegated to an external service, e.g., one that is deployed on cloud computing infrastructure.

The *system and environment state* from the description above represent the sets of parameter values that are required to select the relevant instance of the parametric model of the system, and which the self-adaptive system cannot modify. Using control theory terminology, these are the values of *observable* parameters. For the UUV system from our case study, for example, these are the values of x_i^{old} and r_i for the sensor model from Fig. 1, $1 \leq i \leq n$.

Algorithm 2 RQV with lookahead

```
1: function LOOKAHEAD(modelParams, propIdx)
2:     s ← observable parameter values in modelParams
3:     for all states s' such that L∞(s, s') ≤ ϵ do
4:         modelParams' ← modelParams ⊕ s'
5:         key ← (modelParams', propIdx)
6:         result ← QV(modelParams', propIdx)
7:         if LRUCache is full then
8:             Evict LRUCache entry using the LRU policy
9:         end if
10:        LRUCache.add(key, (result, NOW))
11:    end for
12: end function
```

To identify states that are "close" to an m-dimensional state, we calculate the distance between system states $s = (s_1, s_2, \ldots, s_m)$ and $s' = (s'_1, s'_2, \ldots, s'_m)$ using the normalised Chebyshev distance:

$$L_\infty(s, s') = \max_{1 \le i \le m} \frac{|s_i - s'_i|}{s_i^{\max} - s_i^{\min}}, \tag{5}$$

and consider that states s and s' are "close" if $L_\infty(s, s') \le \epsilon$ for some $\epsilon > 0$. The values $s_i^{\min}$ and $s_i^{\max}$ above represent the minimum and maximum values of state parameter s_i, respectively.

The pseudocode for our lookahead technique, which is used in conjunction with the caching mechanism described in the previous section, is shown in Algorithm 2. The LOOKAHEAD function is invoked whenever spare CPU resources are available immediately after the execution of the VERIFY function from Algorithm 1, and with the same parameters as VERIFY.

4.3 Nearly-Optimal Reconfiguration

Our nearly-optimal reconfiguration technique speeds up the execution of RQV steps by selecting the first valid configuration whose cost/utility is sufficiently close to the best configuration encountered over a sufficiently long period of time. For the UUV system from our case study, for instance, this involves relaxing requirement R3, so that configurations whose cost is nearly optimal are permitted.

Nearly-optimal reconfiguration is capable of reducing both the completion of RQV steps and their CPU and memory usage, and can be used on its own or in conjunction with caching/caching and lookahead. In our use of the technique (Algorithm 3), we keep track of the minimum cost *minCost* and maximum cost *maxCost* associated with all configurations encountered during self-adaptation and, after a *learning period* $T > 0$ has elapsed, we accept valid configurations whose *cost* satisfies the constraint

$$cost \le minCost + \alpha(maxCost - minCost) \tag{6}$$

where $0 < \alpha < 1$ is a parameter that reflects the *lenience* of the technique. Otherwise, i.e., if the "learning" period of length T has not been completed or the *cost* does not satisfies the constraint (6), the configuration is not accepted, and the function NEARLYOPTIMALRECONFIGURATION, which is invoked after each valid configuration identified during an RQV step, returns false.

5. IMPLEMENTATION

We implemented a fully-fledged simulator for the self-adaptive UUV system from our case study, using the open-

Algorithm 3 RQV with nearly-optimal reconfiguration

```
1: minCost ← ∞
2: maxCost ← −∞
3: startTime ← −1

4: function NEARLYOPTIMALRECONFIGURATION(cost)
5:     if startTime = −1 then
6:         startTime ← NOW
7:     end if
8:     minCost ← (cost < minCost)?cost : minCost
9:     maxCost ← (cost > maxCost)?cost : maxCost
10:    if NOW ≥ startTime + T then
11:        return cost ≤ minCost+α(maxCost−minCost)
12:    else
13:        return false
14:    end if
15: end function
```

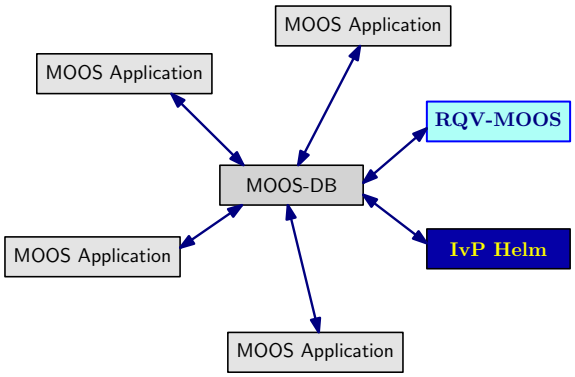

Figure 4: MOOS architecture, adapted from [2]

source MOOS-IvP middleware (`http://oceanai.mit.edu/moos-ivp`) co-developed at MIT and the University of Oxford. MOOS-IvP is a widely used platform for the implementation of autonomous applications on unmanned marine vehicles. The platform is coded in C++, and is typically deployed on the payload computer of an autonomous vehicle, so as to not interfere with the navigation and control system running on the main vehicle computer [2, 3].

The publish-subscribe architecture of the core MOOS software (depicted in Fig. 4) allows applications to publish messages comprising simple key–value pairs with agreed frequencies. These messages can convey, for instance, information about vehicle components monitored by individual applications or about changes to mission objectives (received from a human operator or a peer unmanned vehicle). Any interested "listener" applications can then act upon these messages, e.g., by adjusting the parameters of the navigation and control system they are responsible for.

In addition, user-implemented MOOS applications can propose *behaviours*, i.e., combinations of boolean logic constraints and piecewise-linear utility functions parameterised, for instance, by parameters of the navigation and control system such as *heading*, *speed* or *depth*. A special component of the platform, the IvP Helm, is responsible for the periodic collection and integration of these proposed behaviours. This component uses Interval Programming (IvP) multi-objective optimisation to "reconcile" the behaviours proposed by all contributing applications, and publishes the optimal solution (i.e., an optimal point in the decision space

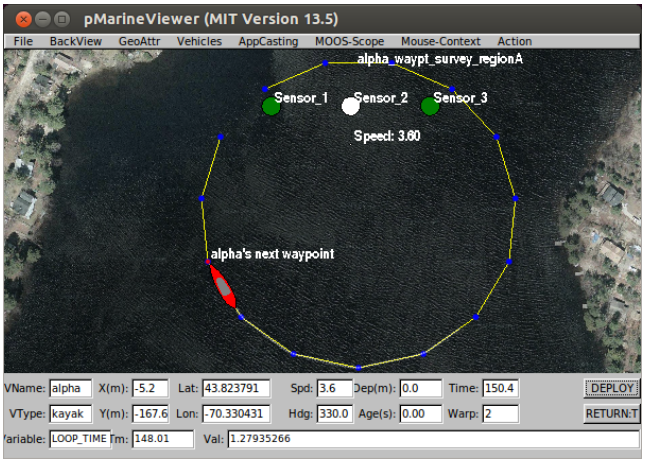

Figure 5: Self-adaptive UUV simulator

defined by the constraints and utility functions) as key–value pairs that the other applications can subscribe to receive.

For our case study from Section 2, we developed a Runtime Quantitative Verification MOOS (RQV-MOOS) application (Fig. 4) that carries out quantitative verification operations using an embedded instance of the PRISM probabilistic model checker [22]. RQV-MOOS operates by:

(a) listening for messages published by the control software for the n sensors, to obtain information about the current rates r_1, r_2, ..., r_n that the sensors operate at;

(b) carrying out periodic RQV steps, to verify the system compliance with the requirements R1–R3 from Section 2, and to select new configurations (i.e., new values for the sensor on/off parameters x_1, x_2, ... x_n and for the UUV speed s);

(c) publishing messages that announce the new sensor configurations, so that the control software for sensor i receives the new x_i, $1 \leq i \leq n$;

(d) proposing a *behaviour* that recommends to the IvP Helm the new UUV speed s.

Fig. 5 shows a screenshot of a 3-sensor instance of our self-adaptive UUV simulator, at a time moment when sensors 1 and 3 are switched on (i.e., $x_1 = x_3 = 1$), sensor 2 is switched off (i.e., $x_2 = 0$), and the UUV speed is $s = 3.6$m/s. The code for our RQV-MOOS application and UUV simulator, and a video recording of the demo from which we extracted the screenshot in Fig. 5 are freely available at `http://www-users.cs.york.ac.uk/~simos/SEAMS`.

6. EVALUATION

To evaluate the effectiveness of RQV augmented with caching, lookahead, nearly-optimal reconfiguration and combinations thereof, we carried out a broad range of experiments using a test suite comprising 24 scenarios. These scenarios considered instances of our self-adaptive UUV system with the different combinations of the following characteristics:

- numbers of sensors $n \in \{3, 4, 6\}$;
- mission duration of 250, 500, 1000 and 2000 RQV steps (executed every 5s);
- level of sensor rate fluctuations *during periods of normal behaviour*—low (up to 2%) and high (up to 10%); as we explain later in this section, sensors also have periods of unexpected behaviour, when the rates change potentially dramatically;
- pattern of sensor failures and/or significant drops in measurement rates.

Table 2: Number of configurations for different system sizes

Number of sensors (n)	Total number of possible configurations
3	35×10^6
4	37×10^8
6	39×10^{13}

For each scenario, separate experiments were carried out using standard RQV, and RQV augmented with caching, lookahead and caching, nearly-optimal configuration, and nearly-optimal configuration and caching. For each technique and combination of techniques that involved the use of caching, we experimented with cache sizes ranging from 10^4 to 10^6 entries. As shown in Table 2, all these cache size are orders of magnitude smaller than the amount of memory required to store the quantitative verification results associated with all possible system configurations.

All experiments were carried out on a standard machine with Inter Core i7-3770 3.40GHZ with 8GB of RAM, running Ubuntu 12.04 64-bit.

We start the presentation of our evaluation with a series of results associated with a 3-sensor self-adaptive UUV whose sensors have the energy-related parameters in Table 1, and are experiencing the pattern of variation in measurement rates depicted in Fig. 6. The adaptation decisions taken by the system for this scenario are explained below (the entries 'A' to 'L' refer to the labels in Fig. 6):

A The rate of sensor 3 decreases significantly, and the UUV switches it off and starts using sensor 2.

B As sensor 2 also experiences a decrease in rate, the UUV continues with only sensor 1, but needs to decrease its speed considerably in order to obtain sufficient measurements per every 10 metres travelled (cf. requirement R1).

C-D Sensor 1 operates with low rate and is switched off; the UUV starts using sensor 3 and reduces its speed.

E Sensor 3 recovers and the UUV starts using it along with sensor 1; the speed is increased accordingly.

F Sensors that are switched off due to poor performance are periodically tested to find out whether they recovered. Since they may not have recovered, none of the other sensor UUV parameters is modified during these tests.

G Sensor 1 is experiencing problems, so the slightly lower-rate sensor 2 takes over alongside sensor 3, with a suitable lowering of the UUV speed.

H Sensor recovery is not always detected immediately.

J With both sensors 2 and 3 in a bad state, the UUV speed must be reduced to a value at which sensor 1 alone can satisfy requirement R1.

K The recovery of sensor 2 enables the UUV to continue the mission at higher speed.

L When sensor 3 recovers too, it is preferred to sensor 2 as it is more energy efficient (cf. Table 1).

Figs. 7 and 8 show the RQV response time (i.e., the time to complete all the quantitative verification operations for an

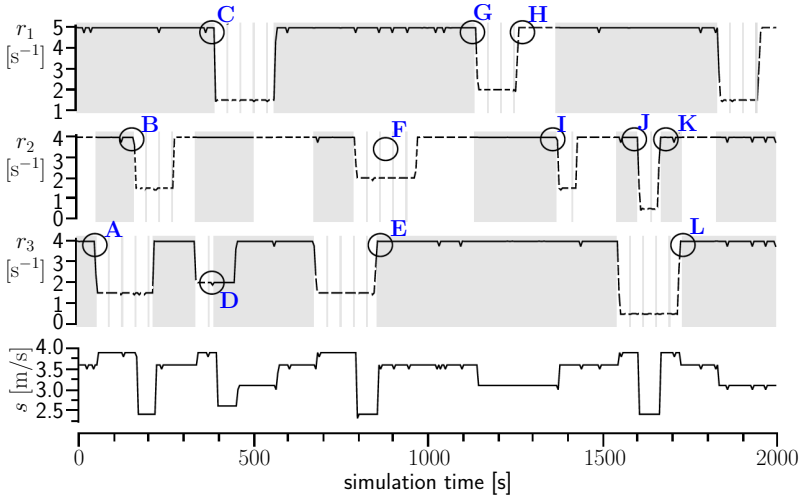

Figure 6: Sample pattern of sensor failures and drops in measurement rates for a 3-sensor system, and the sensor configurations and speed chosen by the self-adaptive UUV (shaded areas correspond to a sensor being switched on, and areas not shaded to the sensor being switched off).

RQV step, averaged over successive sequences of 100 steps) and the cumulated number of quantitative verification operations for a 10,000s simulation of the 3-sensor self-adaptive UUV. The pattern of sensor-rate variation described above corresponds to the first 2,000s of simulated time, and a similar pattern was applied for the remainder of the simulation. In addition to this pattern, the sensor rates for the experiments shown in the two diagrams were also varied during periods when they appear constant in Fig. 6, by a values drawn from a uniform distribution between $[-2\%, 2\%]$ and $[-10\%, 10\%]$ of the maximum rate for these sensors, respectively. Note that the "lookahead" quantitative verification operations, which are carried outside the actual RQV step, are not included in the results reported in Figs. 7–8.

All three techniques and combinations of techniques presented in Figs. 7 achieved very good results for all cache sizes, with an overall number of quantitative verification operations of only 5.2%–7.6% of the number of operations carried out by standard RQV. Clearly, the low sensor-rate variations during periods of normal operation meant that the configurations to be verified were already available in cache or, for the nearly-optimal reconfiguration and caching technique, were similar to configurations seen before. The response time was consistently below 17.5%, with the exception of the initial, "learning" period of the nearly-optimal reconfiguration technique (used with a lenience parameter $\alpha = 0.1$, and yielding new configurations that were on average 28.3% more expensive than the optimal configurations generated by the other approaches). The benefit of a cache larger than 10^4 entries was marginal.

The cache size did make a difference, however, when the sensor-rate variation during normal operation was higher (Fig. 8). In this case, the smaller cache size examined (i.e., 10^4 entries) supported the reduction of the overall number of quantitative verification operations to just 58.6% for the caching, and lookahead and caching techniques. The medium cache size, 10^5 entries, achieved reductions to 34.4% and 30.1% for caching, and lookahead and caching, respectively, showing that the cache was insufficient to support effective lookahead. In contrast, the largest cache size, 10^6 entries, supports lookahead, reducing the number of verification operations to only 17.6% when this technique is used in conjunction with cache, compared to also 34.4% when cache alone was used. Near-optimal reconfiguration continues to work well, providing reductions to 19–23% of the number of operations for standard RQV, irrespective of cache sizes, but at the expense of using suboptimal configurations. Unsurprisingly, the same pattern is observed in the average response times, with the medium sized cache sufficient for the RQV augmented with caching alone, but not for RQV with lookahead and caching.

One final result that we present in this section is an exploration of the scalability of the techniques introduced in this paper. For this purpose, we ran simulations of self-adaptive UUV systems with 3, 4 and 6 sensors, using RQV augmented with the same three techniques and combinations of techniques as above. The resulting average RQV response times, shown in Fig. 9, are given as percentages of the response times for standard RQV, and show that very similar benefits are obtained for across all system sizes. When caching alone is used, response times of between 1% and 22% of those for standard RQV are obtained with a 10^5-entry cache, and no further improvements are possible when the larger cache is used. Lookahead, however, can take advantage of the larger, 10^6-entry cache, to reduce the response time by a further 6% on average, to between approximately 0.5% and 16% of the standard RQV response time. Nearly-optimal reconfiguration with caching is again doing well irrespective of the cache size, although at the expense of an increased overall cost of 28.3% for $n = 3$, 29.7% for $n = 4$ and 34.2% for $n = 6$.

7. RELATED WORK

Caching, lookahead and techniques resembling nearly-optimal reconfiguration have been around for a few decades and have been widely applied in engineering software systems. Caching was extensively used in a range of applications, including web servers [26] and processors [16]. Lookahead-style techniques have previously been applied to dynamic resource provisioning in computer clusters [19] and speech recognition [25]. Variants of suboptimal optimisation/ configuration were widely used in optimisation of network traf-

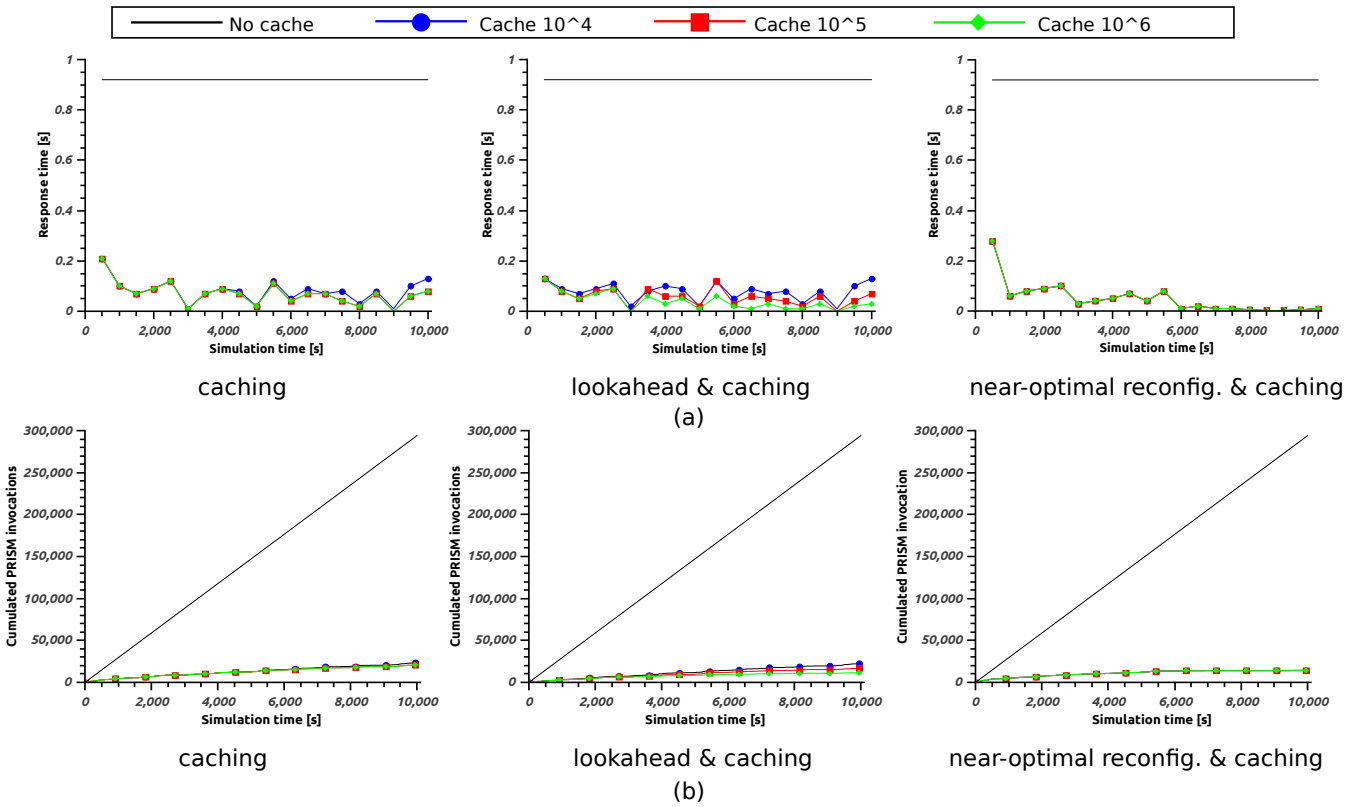

Figure 7: Effect of efficient RQV techniques on (a) the average time required to decide a new configuration during an RQV step (*response times* are averaged over 100 RQV steps); and (b) the total number of quantitative verification operations over 2000 RQV steps, for a scenario with low (i.e., $\leq 2\%$) sensor-rate fluctuation during normal operation.

fic [28], malware detection [15], and many other areas. To the best of our knowledge, however, this is the first work that suggests the use of these techniques in the context of runtime model checking of software systems.

The RQV community has explored ways of improving the RQV efficiency, but the techniques proposed to date complement those introduced in our paper. The approach in [24, 14] exploits the fact that small changes in a Markovian model typically affect a subset of its strongly connected components (SCCs). By applying incremental verification and reusing verification results from the unaffected SCCs, the approach reduces verification overheads. The work in [10, 17] applies compositional and incremental quantitative verification to component-based systems. Each component is verified independently and system-level properties are inferred without generating the complete, monolithic model of the entire system. In [13], reliability-related requirements are pre-computed and formulated as symbolic expressions parameterised with model variables. At runtime, these expressions are evaluated by replacing the variables with values obtained while monitoring the system. The techniques introduced in our paper complement these RQV variants, and can be combined with any of them with limited effort.

8. CONCLUSION

We adapted three software engineering techniques—caching, lookahead and nearly-optimal reconfiguration—for use in improving the response time and reducing the overheads of runtime quantitative verification (RQV). To evaluate the benefits and limitations of these techniques, we developed a self-adaptive unmanned underwater vehicle (UUV) simulator and carried out experiments involving a wide range of realistic scenarios. The experimental results showed that caching can improve the RQV efficiency significantly when there are small variations in the state of the self-adaptive system and its environment during periods of normal operation. When these variations are more significant, lookahead and caching used together achieve much better results, but require a larger cache for this and depend on the availability of spare computation resources. In contrast, near-optimal reconfiguration and caching is less sensitive to cache sizes, and achieves very good performance irrespective of system size, even for high variations in the state of the system during periods of normal operation.

In future work, we are planning to evaluate the techniques presented in this paper in additional applications, including through integration with compositional and incremental quantitative verification [10, 14, 17, 24], with a view to improve RQV efficiency beyond what each of these classes of techniques can achieve on their own. In the longer term, we envisage the development of a collection of adaptive techniques for runtime quantitative verification, which could be used interchangeably or depending on the needs of each RQV step, and the resources available for its completion.

9. ACKNOWLEDGMENTS

This work was partly supported by Dstl grant “Runtime Quantitative Verification of Self-Adaptive AI Systems”. The

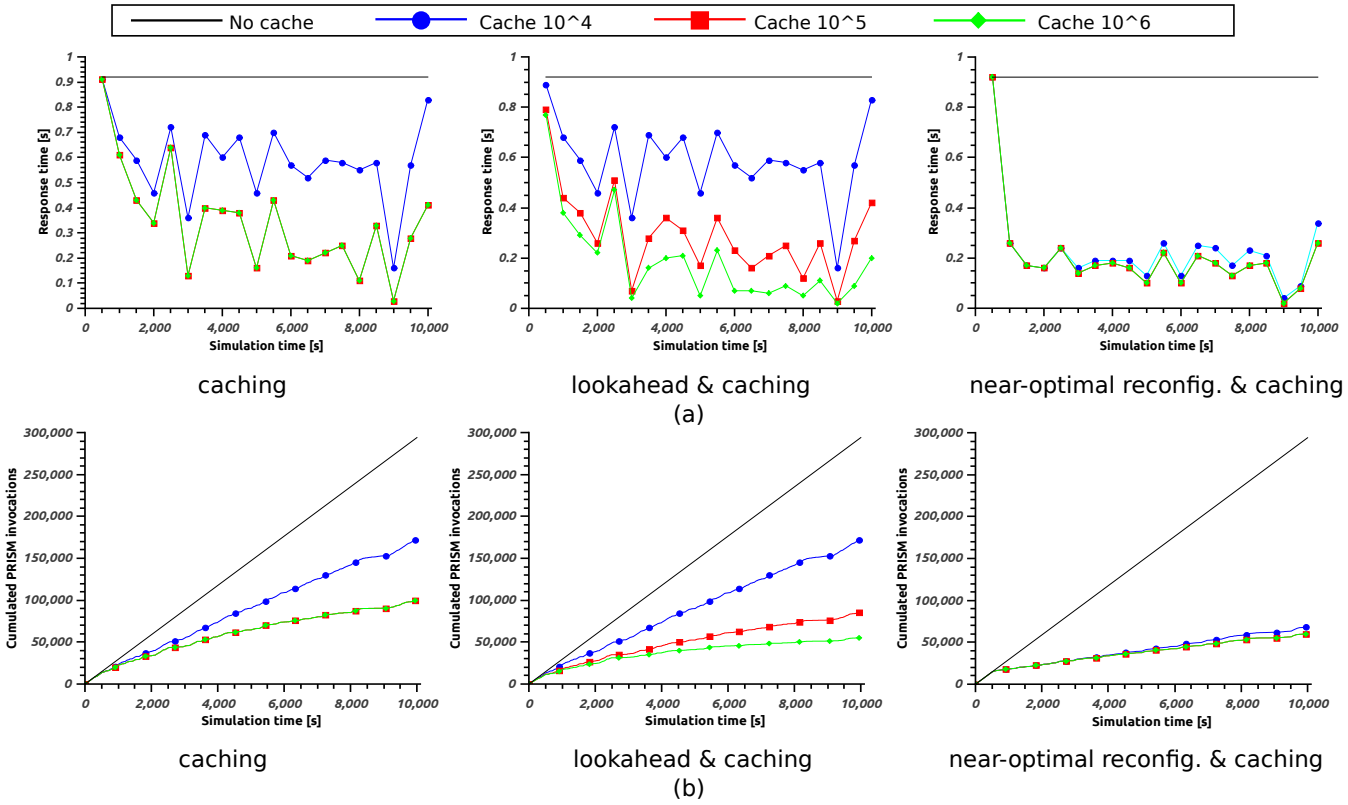

Figure 8: Effect of efficient RQV techniques on (a) the average time required to decide a new configuration during an RQV step (*response times* are averaged over 100 RQV steps); and (b) the total number of quantitative verification operations over 2000 RQV steps, for a scenario with high (i.e., up to 10%) sensor-rate fluctuation during normal operation.

concepts and techniques described in this paper are solely those of the authors, and do not necessarily reflect the views of the funding organisation.

10. REFERENCES

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous-time markov chains. *ACM Trans. Comput. Logic*, 1(1):162–170, July 2000.

[2] M. Benjamin, H. Schmidt, P. Newman, and J. Leonard. Autonomy for unmanned marine vehicles with moos-ivp. In M. L. Seto, editor, *Marine Robot Autonomy*, pages 47–90. Springer, 2013.

[3] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard. Nested Autonomy for Unmanned Marine Vehicles with MOOS-IvP. *Journal of Field Robotics*, 27(6):834–875, 2010.

[4] R. Calinescu. Emerging techniques for the engineering of self-adaptive high-integrity software. In J. Camara et al., editors, *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 297–310. Springer, 2013.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, Sept. 2012.

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Soft. Eng.*, 37(3):387–409, 2011.

[7] R. Calinescu, K. Johnson, and Y. Rafiq. Using observation ageing to improve Markovian model learning in QoS engineering. In *Intl. Conf. on Performance Eng.*, pages 505–510, 2011.

[8] R. Calinescu, K. Johnson, and Y. Rafiq. Developing self-verifying service-based systems. In *Intl. Conf. on Automated Soft. Eng.*, pages 734–737, 2013.

[9] R. Calinescu, S. Kikuchi, and K. Johnson. Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In *Large-Scale Complex IT Systems*, volume 7539 of *LNCS*, pages 303–329. Springer, 2012.

[10] R. Calinescu, S. Kikuchi, and K. Johnson. Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In *Large-Scale Complex IT Systems*, volume 7539 of *LNCS*, pages 303–329. Springer, 2012.

[11] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In *Intl. Conf. on Soft. Eng.*, pages 100–110, 2009.

[12] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *Intl. Conf. on Soft. Eng.*, pages 111–121, 2009.

[13] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Intl. Conf. on Soft. Eng.*, pages 341–350, 2011.

[14] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and

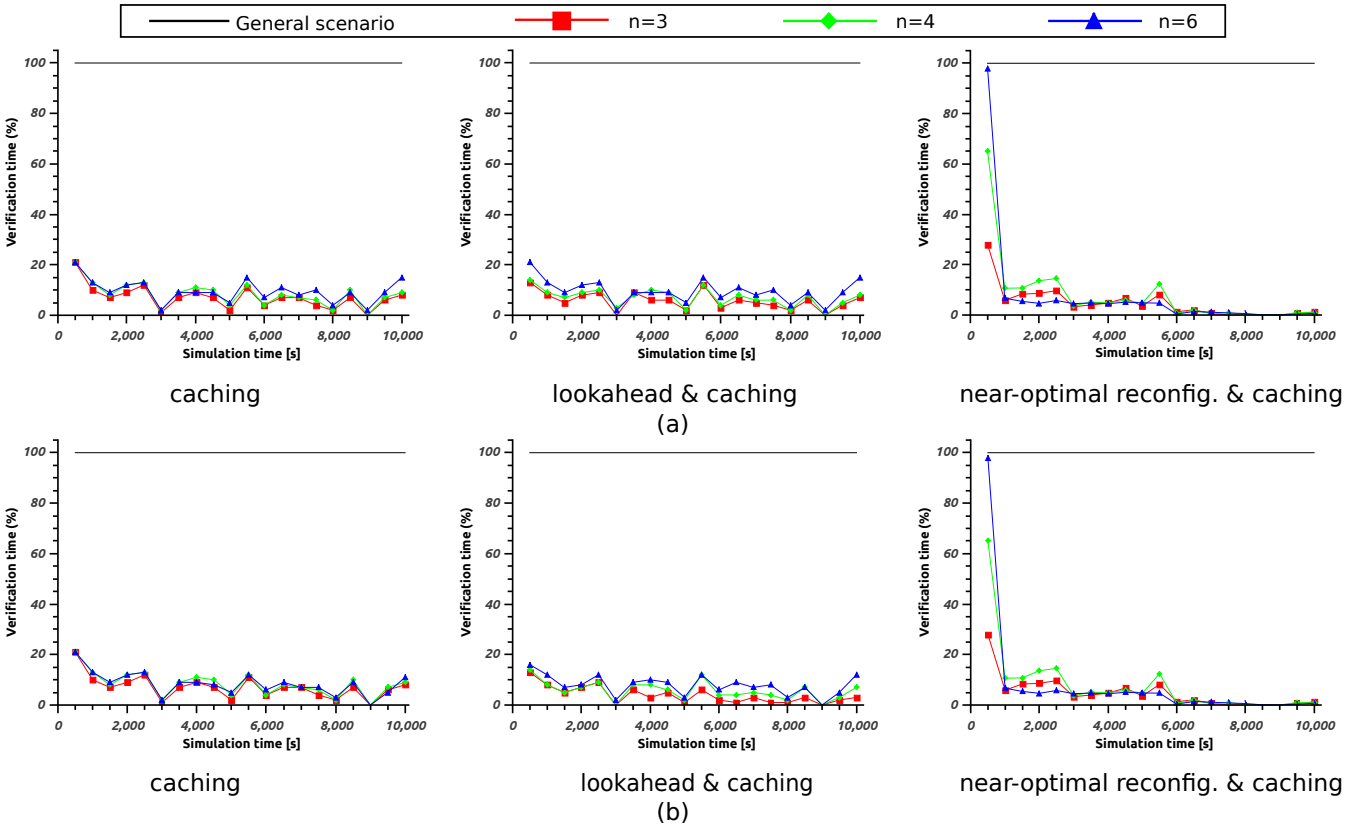

Figure 9: Effect of efficient RQV verification on the response time (averaged over 100 RQV steps) for UUV systems with 3, 4 and 6 sensors, low sensor-rate variation during normal operation periods, and using (a) a 10^5-entry cache; and (b) a 10^6-entry cache.

M. Ujma. Incremental runtime verification of probabilistic systems. In S. Qadeer et al., editors, *Runtime Verification*, volume 7687 of *LNCS*, pages 314–319. Springer, 2013.

[15] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Symp. on Security and Privacy*, pages 45–60, 2010.

[16] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *Intl. Symp. on Comp. Arch.*, pages 124–131, 1983.

[17] K. Johnson, R. Calinescu, and S. Kikuchi. An incremental verification framework for component-based software systems. In *Intl. Symp. on Component-Based Soft. Eng.*, pages 33–42, 2013.

[18] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Eval. of Syst.*, pages 243–244, 2005.

[19] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Comp.*, 12(1):1–15, 2009.

[20] M. Kwiatkowska. Quantitative verification: models, techniques and tools. In *6th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Foundations of Soft. Eng.*, pages 449–458, 2007.

[21] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Intl. Conf. on Formal Methods for Performance Eval.*, pages 220–270, 2007.

[22] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: verification of probabilistic real-time systems. In *Intl. Conf. on Computer Aided Verification*, pages 585–591, 2011.

[23] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza et al., editors, *Intl. Conf. on Tools and Alg. for the Constr. and Analysis of Syst.*, volume 6105 of *LNCS*, pages 23–37, 2010.

[24] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Intl. Conf. on Dependable Syst. Networks*, pages 359–370, 2011.

[25] S. Ortmanns, H. Ney, and A. Eiden. Language-model look-ahead for large vocabulary speech recognition. In *Intl. Conf. on Spoken Lang.*, pages 2095–2098, 1996.

[26] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.

[27] M. Seto, L. Paull, and S. Saeedi. Introduction to autonomy for marine robots. In M. L. Seto, editor, *Marine Robot Autonomy*, pages 1–46. Springer, 2013.

[28] A. Sridharan, R. Guérin, and C. Diot. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. *IEEE/ACM Trans. Netw.*, 13(2):234–247, Apr. 2005.