Article:
Calinescu, Radu Constantin orcid.org/0000-0002-2678-9260, Ceska, Milan, Gerasimou, Simos orcid.org/0000-0002-2706-5272 et al. (2 more authors) (2018) Efficient Synthesis of Robust Models for Stochastic Systems. Journal of Systems and Software. pp. 140-158. ISSN: 0164-1212

https://doi.org/10.1016/j.jss.2018.05.013

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

Efficient Synthesis of Robust Models for Stochastic Systems

Radu Calinescu[1], Milan Češka[2], Simos Gerasimou[1], Marta Kwiatkowska[3], and Nicola Paoletti[4]

1. Department of Computer Science, University of York, UK
2. Faculty of Information Technology, Brno University of Technology, Czech Republic
3. Department of Computer Science, University of Oxford, UK
4. Department of Computer Science, Stony Brook University, USA

Abstract

We describe a tool-supported method for the efficient synthesis of parametric continuous-time Markov chains (pCTMC) that correspond to *robust designs* of a system under development. The pCTMCs generated by our RObust DEsign Synthesis (RODES) method are resilient to changes in the system's operational profile, satisfy strict reliability, performance and other quality constraints, and are Pareto-optimal or nearly Pareto-optimal with respect to a set of quality optimisation criteria. By integrating sensitivity analysis at designer-specified tolerance levels and Pareto optimality, RODES produces designs that are potentially slightly suboptimal in return for less sensitivity—an acceptable trade-off in engineering practice. We demonstrate the effectiveness of our method and the efficiency of its GPU-accelerated tool support across multiple application domains by using RODES to design a producer-consumer system, a replicated file system and a workstation cluster system.

Keywords: software performance and reliability engineering; probabilistic model synthesis; multi-objective optimisation; robust design

1. Introduction

Robustness is a key characteristic of both natural [1] and human-made [2] systems. Systems that cannot tolerate change are prone to frequent failures and require regular maintenance. As such, engineering disciplines like mechanical and electrical engineering treat robustness as a first-class citizen by designing their systems based on established tolerance standards (e.g. [3, 4]). By comparison, software engineering is lagging far behind. Despite significant advances in software performance and reliability engineering [5, 6, 7, 8, 9, 10], the quality attributes of software systems are typically analysed for point estimates of stochastic system parameters such as component service rates or failure probabilities. Even the techniques that assess the sensitivity of quality attributes to parameter changes (e.g. [11, 12, 13, 14, 15]) focus on the analysis of a given design at a time instead of systematically designing robustness into the system under development (SUD).

To address these limitations, we propose a tool-supported method for the efficient synthesis of parametric continuous-time Markov chains (pCTMCs) that correspond to robust SUD designs. Our RObust DEsign Synthesis (RODES) method generates sets of pCTMCs that:

(i) are resilient to pre-specified *tolerances* in the SUD parameters, i.e., to changes in the SUD's operational profile;

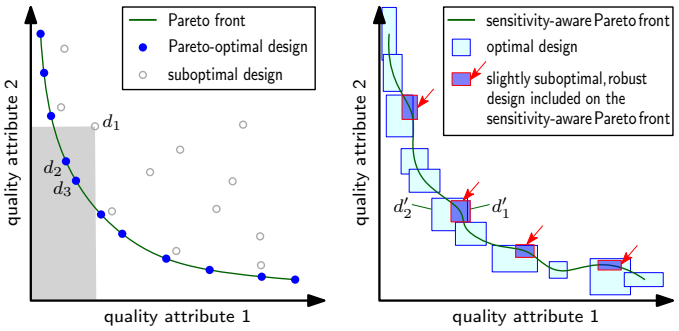
(ii) satisfy strict performance, reliability and other quality constraints;

(iii) are Pareto-optimal or nearly Pareto optimal with respect to a set of quality optimisation criteria.

RODES comprises two steps. In the first step, the SUD design space is modelled as a pCTMC with discrete and continuous parameters corresponding to alternative system architectures and to ranges of possible values for the SUD parameters, respectively. In the second step, a multi-objective optimisation technique is used to obtain a set of low-sensitivity, Pareto-optimal or nearly Pareto-optimal SUD designs by fixing the discrete parameters (thus selecting specific architectures) and restricting the continuous parameters to bounded intervals that reflect the pre-specified tolerances. The designs that are slightly suboptimal have the advantage of a lower sensitivity than the optimal designs with similar quality attributes, achieving a beneficial compromise between optimality and sensitivity. A *sensitivity-aware Pareto dominance relation* is introduced in the paper to formally capture this trade-off.

Figure 1 shows the differences between a traditional Pareto front, which corresponds to a fixed SUD operational profile, and a sensitivity-aware Pareto front generated by RODES, which corresponds to a SUD operational profile that can change within pre-specified bounds. Accordingly, the designs from the RODES sensitivity-aware Pareto front are bounded regions of quality-attribute values for the system. The size and shape of these regions convey the sensitivity of the synthesised designs to parameter changes within the pre-specified tolerances. Small quality-attribute regions correspond to particularly robust

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at
https://doi.org/10.1016/j.jss.2018.05.013



(a) Traditional Pareto front: design d_1 is suboptimal because designs d_2 and d_3 from the shaded area have better (i.e., lower) quality attributes than d_1

(b) Sensitivity-aware Pareto front: the slightly suboptimal design d'_1 belongs to the front because it has a much lower sensitivity than optimal design d'_2

Figure 1: Traditional Pareto front (a) versus sensitivity-aware Pareto front (b) for two quality attributes that require minimisation (e.g., response time and probability of failure).

designs that cope with variations in the system parameters without exposing users to significant changes in quality attributes. These designs require reduced maintenance, and can be implemented using high-variability components that are cheaper to develop or obtain off-the-shelf than low-variability components. Large quality-attribute regions from a RODES Pareto front—while still the most robust for the quality attribute trade-offs they correspond to—are associated with designs that are sensitive to SUD parameters variations. These designs may involve high maintenance and/or development costs, so they should only be used if justified by their other characteristics (e.g. desirable quality attribute trade-offs).

To the best of our knowledge, RODES is the first solution that integrates multi-objective stochastic model synthesis and sensitivity analysis into an end-to-end, tool-supported design method. As we show in detail in Section 7, the existing research addresses the challenges associated with design synthesis (e.g. [16, 17]) and sensitivity analysis (e.g. [11, 12, 13, 14, 15]) separately. The main contributions of our paper are:

1. The extension of the notion of *parameter tolerance* from other engineering disciplines for application to software architecture.
2. The definitions of the parametric Markov chain synthesis problem and of the sensitivity-aware Pareto dominance relation for the synthesis of robust models for stochastic systems.
3. The RODES method for the generation of sensitivity-aware Pareto fronts by integrating multi-objective probabilistic model synthesis and precise *p*CTMC parameter synthesis.
4. A GPU-accelerated tool that implements the RODES method and is available preinstalled on an easy-to-use VirtualBox instance from our project website `https://github.com/gerasimou/RODES/wiki`.
5. A repository of case studies demonstrating the successful application of RODES to a replicated file system used by Google's search engine, a cluster availability management system, and a producer-consumer system.

These contributions significantly extend our conference paper on robust model synthesis [18] and the prototype probabilistic model synthesis tool [19] in several ways. First, we provide a more detailed description of our solution, including a running example and new experimental results. Second, we greatly improve the scalability of RODES by integrating the GPU-accelerated analysis of candidate designs into our prototype tool [19]. Third, we extend the experimental evaluation to demonstrate the impact of the GPU acceleration. Finally, we present an additional case study in which we apply RODES to a producer-consumer system, and we use the systems and models from our experiments to assemble a repository of case studies available on our project website.

The remainder of the paper is organised as follows. Section 2 introduces the RODES design-space modelling language and the formalism to specify quality constraints and optimisation criteria. Section 3 defines the sensitivity-aware dominance relation and introduces the parametric Markov chain synthesis problem. We then present our method for synthesising robust designs in the form of a sensitivity-aware Pareto set, and the GPU-accelerated tool RODES implementing the method in Sections 4 and 5, respectively. Finally, we evaluate our method within three case studies in Section 6, discuss related work in Section 7, and conclude the paper with a summary and future work in Section 8.

2. Modelling and Specification Language for Probabilistic Systems

This section formalises three key elements underpinning the formulation of the robust design problem: 1) the modelling of the design space of a SUD, 2) the specification of quality attributes and requirements, and 3) the sensitivity of a design.

2.1. Design space modelling

We use a *parametric continuous-time Markov chain* (*p*CTMC) to define the design space of a SUD. To this end, we extend the original *p*CTMC definition [20], where only real-valued parameters determining the transition rates of the Markov chain are considered, and assume that a *p*CTMC also includes discrete parameters affecting its state space. Our definition captures the need for both discrete parameters encoding architectural structural information (e.g. by selecting between alternative implementations of a software component) and continuous parameters encoding configurable aspects of the system (e.g. network latency or throughput). As such, a candidate system design corresponds to a fixed discrete parameter valuation and to continuous parameter values from a (small) region.

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at
https://doi.org/10.1016/j.jss.2018.05.013

Definition 1 (pCTMC)**.** *Let K be a finite set of real-valued parameters such that the domain of each parameter $k \in K$ is a closed interval $[k^\perp, k^\top] \subset \mathbb{R}$, and D a finite set of discrete parameters such that the domain of each parameter $d \in D$ is a set $T^d \subset \mathbb{Z}$. Let also $\mathcal{P} = \bigtimes_{k \in K} [k^\perp, k^\top]$ and $\mathcal{Q} = \bigtimes_{d \in D} T^d$ be the continuous and the discrete* parameter spaces *induced by K and D, respectively. A pCTMC over K and D is a tuple*

$$\mathcal{C}(\mathcal{P}, \mathcal{Q}) = (\mathcal{D}_S, \mathcal{D}_{init}, \mathcal{D}_{\mathbf{R}}, L), \tag{1}$$

where, for any discrete parameter valuation $q \in \mathcal{Q}$:

- *$\mathcal{D}_S(q) = S$ is a finite set of* states*, and $\mathcal{D}_{init}(q) \in S$ is the initial state;*
- *$\mathcal{D}_{\mathbf{R}}(q) : S \times S \to \mathbb{R}[K]$ is a* parametric rate matrix*, where $\mathbb{R}[K]$ denotes the set of polynomials over the reals with variables in K;*
- *$L(q) : S \to 2^{AP}$ is a* labelling function *mapping each state $s \in S$ to the set $L(q)(s) \subseteq AP$ of atomic propositions that hold true in s.*

A pCTMC $C(\mathcal{P}, \mathcal{Q})$ describes the uncountable set of continuous-time Markov chains (CTMCs) $\{\mathcal{C}(p, q) \mid p \in \mathcal{P} \wedge q \in \mathcal{Q}\}$, where each $\mathcal{C}(p, q) = (\mathcal{D}_S(q), \mathcal{D}_{init}(q), \mathbf{R}(p, q), L(q))$ is the instantiated CTMC with transition matrix $\mathbf{R}(p, q)$ obtained by replacing the real-valued parameters in $\mathcal{D}_{\mathbf{R}}(q)$ with their valuation in p.

In our approach we operate with pCTMCs expressed in a high-level modelling language extending the PRISM language [21] which models a system as the parallel composition of a set of *modules*. The state of a module is encoded by a set of finite-range local variables, and its state transitions are defined by probabilistic guarded commands that change these variables, and have the general form:

$$\text{[action] guard } \to e_1 : update_1 + \ldots + e_n : update_n \tag{2}$$

In this command, *guard* is a Boolean expression over all model variables. If the guard evaluates to true, the arithmetic expression e_i, $1 \leq i \leq n$, gives the rate with which the $update_i$ change of the module variables occurs. When *action* is present, all modules comprising commands with this action have to synchronise (i.e., to carry out one of these commands simultaneously) and the resulting rate of such synchronised commands is equal to the multiplication of the individual command rates. Atomic propositions are encoded with label expressions of the form:

$$\text{label "id"} = b \tag{3}$$

where id is a string that identifies the atomic proposition and b is a Boolean expression over the state variables.

We extend the PRISM language with the following constructs (adopted from [16]) for specifying the parameters $k \in K$ and $d \in D$ from Definition 1:

$$\begin{array}{l}\text{evolve double } k\ [min..max] \\ \text{evolve int } d\ [min..max] \\ \text{evolve module } ComponentName\end{array} \tag{4}$$

where $N > 1$ instances of the last construct (with the same component name) define N alternative architectures for a component, introducing the index (between 1 and N) of the selected architecture as an implicit discrete parameter.

As per Definition 1, continuous parameters can only appear in the transition rates (expressions $e_1, \ldots, e_n$ above). Explicit discrete variables (declared using evolve int) can instead appear in any type-consistent expression.

The translation of models expressed in the extended PRISM language into the corresponding pCTMC is fully automatic and follows the probabilistic guarded command semantics described above. The discrete state space $\mathcal{Q}$ results from all possible valuations of explicit discrete variables and implicit discrete variables (different implementations of a module). For a fixed valuation $q \in \mathcal{Q}$, the parametric PRISM model describes a fixed set of modules with a fixed set of finite-range variables, and thus the state space $\mathcal{D}_S(q)$ is given by the Cartesian product of the value ranges for these variables. In contrast, q determines also the parametric rate matrix $\mathcal{D}_{\mathbf{R}}(q)$ and atomic propositions $L(q)$, as q can affect guards and updates of PRISM commands, as well as label expressions.

Example 1 (Producer-consumer model)**.** *As a running example, we consider a simple producer-consumer system with a two-way buffering, illustrated in Figure 2. The pCTMC PRISM model, extended with the evolvable constructs from Definition 4 is shown in Figure 3. The system comprises a producer generating requests with rate* p_rate*. Each request is being transferred to a consumer either via a slow buffer or via a fast buffer with probabilities 0.6 and 0.4, respectively (lines 14 and 15 in Figure 3). The fast buffer transmits requests to the consumer faster than the slow buffer, but it has smaller capacity and is less reliable, as it loses packets with a 5% probability (line 20).*

We consider two alternative designs of the producer-consumer model that differ in the way that the two buffers manage the pending requests. More specifically we consider

1. *a no-redirection design in which once a request is sent to either buffer, the packet is transmitted by that buffer to the consumer (lines 9-22);*
2. *a redirection design that enables the slow buffer to transmit requests to the fast buffer with a probability proportional to its occupancy (lines 23-27). In particular, redirection is disabled when the slow buffer is empty and has maximum rate when it is full and is equal to* s_rate*/10, where* s_rate *is the request transmission rate without redirection.*

In addition to these two alternative designs, the model has two continuous parameters, the packet transmission rate for the slow buffer, r_slow_rate*, and* delta_rate*, i.e. the transmission rate difference between fast and slow buffers. Notably, the rate of packet loss by the fast buffer is proportional to its transmission rate, meaning that the buffer becomes less reliable as its rate increases.*

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

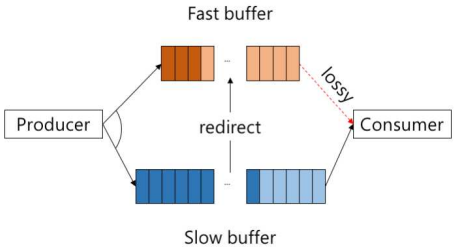


Figure 2: Two-way producer-consumer system.

We formally capture the above system model with its continuous parameters and alternative designs by a pCTMC $\mathcal{C}_{PC}(\mathcal{P},\mathcal{Q})$*, where* $\mathcal{P}=[5,30]\times[0,30]$ *defines the domains for the continuous parameters* r_slow_rate*, and* delta_rate*, respectively, and* $\mathcal{Q}=\{1,2\}$ *defines the domain for the discrete parameter corresponding to the two alternative designs (i.e. modules).*

Definition 2 (Candidate design)**.** *A candidate design of the pCTMC* $\mathcal{C}(\mathcal{P},\mathcal{Q})$ *from (1) is a pCTMC*

$$\mathcal{C}(\mathcal{P}',\{q\})=(\mathcal{D}'_S,\mathcal{D}'_{init},\mathcal{D}'_{\mathbf{R}},L') \tag{5}$$

where $\mathcal{P}'=\bigtimes_{k\in K}[k'^{\perp},k'^{\top}]\subseteq\mathcal{P}$*,* $q\in\mathcal{Q}$*,* $\mathcal{D}'_S(q)=\mathcal{D}_S(q)$*,* $\mathcal{D}'_{\mathbf{R}}(q)=\mathcal{D}_{\mathbf{R}}(q)$*,* $\mathcal{D}'_{init}(q)=\mathcal{D}_{init}(q)$ *and* $L'(q)=L(q)$*. The* tolerance *of the candidate design with respect to the real-valued parameter* $k\in K$ *is defined as*

$$\gamma_k=\frac{k'^{\top}-k'^{\perp}}{2(k^{\top}-k^{\perp})}, \tag{6}$$

in line with the fact that the design restricts the value domain of k *to the interval* $\left[\overline{k}-\gamma_k(k^{\top}-k^{\perp}),\overline{k}+\gamma_k(k^{\top}-k^{\perp})\right]$*,* $\overline{k}=\frac{k'^{\perp}+k'^{\top}}{2}$*.*[1] *For convenience, we will use the shorthand notation* $\mathcal{C}(\mathcal{P}',q)\equiv\mathcal{C}(\mathcal{P}',\{q\})$ *in the rest of the paper.*

Example 2 (Candidate design)**.** *Consider the pCTMC* $C_{PC}(\mathcal{P},\mathcal{Q})$ *from Example 1 and a single tolerance value* $\gamma=0.005$ *for both continuous parameters* r_slow_rate *and* delta_rate*. By (6), candidate designs have continuous parameter ranges of size* $2\gamma(k^{\top}-k^{\perp})=0.25$ *for* r_slow_rate *and of size* 0.3 *for* delta_rate*. Two examples of valid candidate designs for the second module (redirection), obtained using our RODES synthesis method (see also results in Figure 7), are pCTMCs* $d_1=\mathcal{C}_{PC}(\mathcal{P}',2)$ *and* $d_2=\mathcal{C}_{PC}(\mathcal{P}'',2)$ *where* $\mathcal{P}'=[15.02,15.27]\times[1.93,2.23]$*,* $\mathcal{P}''=[13.2,13.45]\times[3.51,3.81]$*. The pCTMCs* $d_3=\mathcal{C}_{PC}(\mathcal{P}''',1)$ *with* $\mathcal{P}'''=[17.24,17.49]\times[2.78,3.08]$ *is instead a valid candidate design for the first module (no redirection).*

[1] In other words, the tolerance of parameter k, γ_k, measures the extent to which k can be perturbed from its reference (midpoint) value.

```
1  ctmc
   // buffer capacities
2  const int slow_max = 31;
3  const int fast_max = 21;
4  cons double p_rate = 40;          //request production rate
5  const double s_rate = 30;         //request transmission rate
   // packet transmission rates
   // trans. rate for the fast buffer is r_slow_rate+delta_rate
6  evolve double r_slow_rate [5..30]; //slow buffer
7  evolve double delta_rate [0..30]; //fast buffer
8  const double c_rate = 40;         //packet consumption rate

// no redirection
9  evolve module Buffer
     // is request sent to fast/slow buffer?
10   fast : [0..1] init 0;
11   slow : [0..1] init 0;
     // buffers
12   buffer_s : [0..slow_max] init 0;
13   buffer_f : [0..fast_max] init 0;
     // has consumer received the packet?
13   consumer : [0..1] init 0;
     //produce
14   [] (fast=0) → p_rate*0.4 : (fast'=1);
15   [] (slow=0) → p_rate*0.6 : (slow'=1);
     //send
16   [] (slow=1) & (buffer_s<slow_max) → s_rate :
                                  (slow'=0) & (buffer_s' = buffer_s +1);
17   [] (fast=1) & (buffer_f<fast_max) → s_rate :
                                  (fast'=0) & (buffer_f' = buffer_f +1);
     //receive
18   [] (consumer=0) & (buffer_s > 0) → r_slow_rate :
                                  (consumer'=1) & (buffer_s' = buffer_s - 1);
19   [] (consumer=0) & (buffer_f > 0) → (r_slow_rate+delta_rate)*0.95 :
                                  (consumer'=1) & (buffer_f' = buffer_f - 1);
     // fast buffer loses the packet
20   [lost] (consumer=0) & (buffer_f > 0) → (r_slow_rate+delta_rate)*0.05 :
                                  (buffer_f' = buffer_f - 1);
     //consume
21   [consume] (consumer=1) → c_rate : (consumer'=0);
22 endmodule

// redirection
23 evolve module Buffer
     ⋮
     //send
24   [] (slow=1) & (buffer_s<slow_max) → s_rate *(1-buffer_s/(10.0*slow_max)) :
                                  (slow'=0) & (buffer_s' = buffer_s +1);
25   [] (slow=1) & (buffer_s>0) & (buffer_f<fast_max) →
                                  s_rate *buffer_s/(10.0*slow_max) :
                                  (slow'=0) & (buffer_f' = buffer_f +1);
26   [] (fast=1) & (buffer_f<fast_max) →
                                  s_rate : (fast'=0) & (buffer_f' = buffer_f +1);
     ⋮
27 endmodule
```

Figure 3: PRISM-RODES encoding of pCTMC model of a producer-consumer system with two-way buffering and redirection. In the second module only the commands that differ from the first module are reported.

2.2. Quality attribute specification and requirements

We specify quality attributes over pCTMCs-defined design spaces using *continuous stochastic logic* (CSL) extended with reward operators [22]. Our focus is on timed properties of pCTMCs expressed by the time-bounded fragment of CSL with rewards comprising state formulae (Φ) and path formulae (ϕ) with the syntax:

$$\begin{array}{l}\Phi ::= \mathsf{true} \mid a \mid \neg\Phi \mid \Phi\wedge\Phi \mid P_{\sim r}[\phi] \mid R_{\sim r}[C^{\leq t}]\\ \phi ::= X\ \Phi \mid \Phi\ U^{I}\Phi\end{array}, \tag{7}$$

where a is an atomic proposition evaluated over states, $\sim\in\{<,\leq,\geq,>\}$ is a relational operator, r is a probability

($r \in [0,1]$) or reward ($r \in \mathbb{R}_{\geq 0}$) threshold[2], $t \in \mathbb{R}_{\geq 0}$ is a time bound, and $I \subseteq \mathbb{R}_{\geq 0}$ is a bounded time interval. The 'future' operator, F, and 'globally' operator, G, are derived from U in the standard way[3]. As briefly discussed in Section 4.2, our approach can be extended to unbounded CSL.

Traditionally, the CSL semantics is defined for CTMCs using a satisfaction relation $\vDash$. Intuitively, a state $s \vDash P_{\sim r}[\phi]$ iff the probability of the set of paths starting in s and satisfying ϕ meets $\sim r$. A path $\omega = s_0 t_0 s_1 t_1 \ldots$ satisfies the formula $\Phi\ U^I\ \Psi$ iff there exists a time $t \in I$ such that $(\omega@t \vDash \Psi \wedge \forall t' \in [0,t).\omega@t' \vDash \Phi)$, where $\omega@t$ denotes the state in ω at time t. A state $s \vDash R_{\sim r}[C^{\leq t}]$ iff the expected rewards over the path starting in s and cumulated within t time units satisfies $\sim r$, where the rates with which reward is acquired in each state and the reward acquired at each transition are defined by a *reward structure*.

In line with our previous work [23], we introduce a *satisfaction function* $\Lambda_\phi : \mathcal{P} \times \mathcal{Q} \to [0,1]$ that quantifies how the satisfaction probability associated with a path CSL formula ϕ relates to the parameters of a pCTMC $\mathcal{C}(\mathcal{P}, \mathcal{Q})$, where, for any $(p,q) \in \mathcal{P} \times \mathcal{Q}$, $\Lambda_\phi(p,q)$ is the probability that ϕ is satisfied by the set of paths from the initial state $\mathcal{D}_{init}(q)$ of the instantiated CTMC $\mathcal{C}(p,q)$. The satisfaction function for reward CSL formulae is defined analogously.

Quality requirements. We assume that the quality requirements of a SUD with design space given by a pCTMC $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ are defined in terms of:

1) A finite set of objective functions $\{f_i\}_{i \in I}$ corresponding to quality attributes of the system and defined in terms of a set of CSL path formulas $\{\phi_i\}_{i \in I}$, such that for any $i \in I$ and $(p,q) \in \mathcal{P} \times \mathcal{Q}$,

$$f_i(\mathcal{C}(p,q)) = \Lambda_{\phi_i}(p,q); \tag{8}$$

2) A finite set of Boolean constraints $\{c_j\}_{j \in J}$ corresponding to the set of CSL path formulas $\{\psi_j\}_{j \in J}$ and thresholds $\{\sim_j r_j\}_{j \in J}$, such that for any $j \in J$ and $(p,q) \in \mathcal{P} \times \mathcal{Q}$,

$$c_j(\mathcal{C}(p,q)) \Leftrightarrow \Lambda_{\psi_j}(p,q) \sim_j r_j. \tag{9}$$

Note that quality requirements (8) and (9) are defined over (non-parametric) CTMCs, but, in order to compare candidate designs with respect to some objective function, we need to interpret quality requirements over pCTMCs. Indeed, due to the continuous parameter space, a single candidate design induces an infinite number of objective function values, from which the designer must choose a representative value. For a candidate design $\mathcal{C}(\mathcal{P}', q)$ and objective f_i, this is typically identified as one of the minimum, maximum and mid-range value of $f_i(\mathcal{C}(p,q))$ over all $p \in \mathcal{P}'$, as illustrated in Table 1.

Table 1: Alternative definitions for objective functions $\{f_i\}_{i \in I}$ over candidate designs.

Type	Notation	Definition
lower bound	$f_i^{\perp}(\mathcal{C}(\mathcal{P}', q))$	$\inf_{p \in \mathcal{P}'} \Lambda_{\phi_i}(p,q)$
upper bound	$f_i^{\top}(\mathcal{C}(\mathcal{P}', q))$	$\sup_{p \in \mathcal{P}'} \Lambda_{\phi_i}(p,q)$
mid-range	$f_i^{\bullet}(\mathcal{C}(\mathcal{P}', q))$	$(f_i^{\perp}(\mathcal{C}(\mathcal{P}', q)) + f_i^{\top}(\mathcal{C}(\mathcal{P}', q)))/2$

On the other hand, constraints have a unique interpretation because they must be met for any parameter value of a candidate design. Formally, for candidate design $\mathcal{C}(\mathcal{P}', q)$ and constraint c_j, we define

$$c_j(\mathcal{C}(\mathcal{P}', q)) \Leftrightarrow \forall p \in \mathcal{P}'.\ c_j(\mathcal{C}(p,q)).$$

Without loss of generality, we will assume that all objective functions $\{f_i\}_{i \in I}$ in Sections 3 and 4 should be minimised and that all thresholds $\{\sim_j r_j\}_{j \in J}$ are upper bounds of the form of $\leq r_j$.

Example 3 (Quality requirements)**.** *Below we define quality requirements for the producer-consumer model of Example 1. We consider two maximisation objectives and one constraint:*

- f_1: $R\{\text{"}consume\text{"}\}_{=?}\ [C^{<=25}]$, *a cumulative transition reward describing the number of requests transferred to the consumer within 25 time units (line 21 in Figure 3);*
- f_2: $P_{=?}\ [G[20,25]((\mathsf{buffer\_s} \geq \mathsf{slow\_max}/2)\&(\mathsf{buffer\_f} \geq \mathsf{fast\_max}/2))]$, *which calculates the probability that the utilisation of both buffers is at least 50% of their respective capacities;*
- c_1: $R\{\text{"}lost\text{"}\}_{\leq 10}\ [C^{<=25}]$, *a cumulative transition reward that limits the number of packets lost within 25 time units (line 20 in Figure 3).*

With these quality requirements, we seek to maximise the system throughput (objective f_1), expressed as the number of requests transferred to the consumer, and also to maximize the probability that both buffers are sufficiently utilised after an initial period (objective f_2). Finally, constraint c_1 imposes a reliability requirement by restricting the number of packets lost to be less than 10 within 25 time units of operation.

2.3. Sensitivity of candidate designs

Quantifying the sensitivity of candidate designs is a crucial step in our robust synthesis method. Intuitively, the sensitivity of a design $\mathcal{C}(\mathcal{P}', q)$ captures how the objective functions $\{f_i\}_{i \in I}$ change in response to variations in the continuous parameters $k \in K$. The variation of each objective f_i is measured by the length of the interval $[f_i^{\perp}(\mathcal{C}(\mathcal{P}', q)),$

[2] For simplicity, we use $\sim r$ to denote the threshold for both probability and reward quality attributes.

[3] $P_{\sim r}[F^I \Phi] = P_{\sim r}[\mathsf{true}\ U^I \Phi]$ and $P_{\sim r}[G^I \Phi] = P_{\approx 1-r}[F^I \neg \Phi]$

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at
https://doi.org/10.1016/j.jss.2018.05.013

$f_i^{\top}(\mathcal{C}(\mathcal{P}',q))]$, describing the range of admissible values for f_i and $\mathcal{C}(\mathcal{P}',q)$ (cf. Table 1). The degree of variation for multiple objectives is given by the product of interval lengths, i.e., the volume of the corresponding quality-attribute region. The sensitivity takes also into account the size of the underlying parameter region, in order to account for designs with different tolerance values. For instance, a design with a large quality-attribute volume and high tolerance (large parameter region volume) must be considered *more robust* (less sensitive) than another design with comparable quality-attribute volume but lower tolerance.

Definition 3 (Sensitivity)**.** *For a set of objective functions $\{f_i\}_{i\in I}$ and tolerances $\{\gamma_k\}_{k\in K}$, the* sensitivity *of a feasible design $\mathcal{C}(\mathcal{P}',q)$ is defined as the volume of its quality-attribute region over the volume of $\mathcal{P}'$:*

$$sens\left(\mathcal{C}(\mathcal{P}',q)\right)=\frac{\prod_{i\in I}\left(f_i^{\top}(\mathcal{C}(\mathcal{P}',q))-f_i^{\perp}(\mathcal{C}(\mathcal{P}',q))\right)}{\prod_{k\in K}2\gamma_k(k^{\top}-k^{\perp})}. \quad (10)$$

Example 4 (Sensitivity)**.** *Consider the candidate designs d_1, d_2, d_3 with tolerance $\gamma = 0.005$ from Example 2, and the objective functions f_1 (number of "consumed" packets) and f_2 (probability of buffers being sufficiently used) introduced in Example 3. Assume the following ranges for f_1 and f_2:*

$$\begin{aligned}
[f_1^{\perp}(d_1), f_1^{\top}(d_1)] &= [416.94, 439.65]\\
[f_2^{\perp}(d_1), f_2^{\top}(d_1)] &= [0.8977, 0.9809]\\
[f_1^{\perp}(d_2), f_1^{\top}(d_2)] &= [407.11, 423.10]\\
[f_2^{\perp}(d_2), f_2^{\top}(d_2)] &= [0.891, 0.9621]\\
[f_1^{\perp}(d_3), f_1^{\top}(d_3)] &= [384.81, 413.09]\\
[f_2^{\perp}(d_3), f_2^{\top}(d_3)] &= [0.7501, 0.8225].
\end{aligned}$$

Recall that the three designs have the same tolerance, thus yielding the same parameter region volume

$$\prod_{k\in K} 2\gamma_k(k^{\top}-k^{\perp}) = 0.25\cdot 0.3 = 0.075$$

The resulting sensitivities are:

$$\begin{aligned}
sens(d_1) &= (439.65-416.94)(0.9809-0.8977)/0.075 = 25.19\\
sens(d_2) &= (423.10-407.11)(0.9621-0.891)/0.075 = 15.16\\
sens(d_3) &= (413.09-384.81)(0.8225-0.7501)/0.075 = 27.3
\end{aligned}$$

indicating that d_2 is the most robust design (with the smallest sensitivity value). The three designs can be visualised in the quality-attribute space *(i.e. the objective space), as shown in Figure 4, providing a direct and intuitive way to assess robustness.*

3. Sensitivity-Aware Pareto Dominance Relation

In this section, we introduce a novel dominance relation that adequately captures tradeoffs between the sensitivity and optimality of candidate designs with respect to given quality requirements, and that enables to formulate the robust design problem as an optimisation problem.

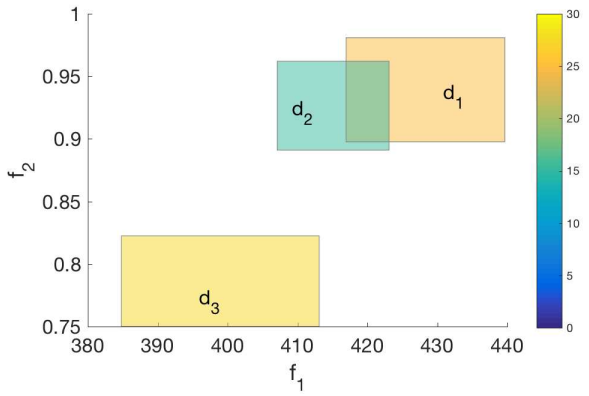


Figure 4: Candidate designs of Example 4 represented in the quality-attribute space and coloured by sensitivity. Designs d_1 and d_2 were synthesised using RODES (full results are reported in Figure 7 on a different scale).

Consider a system with design space $\mathcal{C}(\mathcal{P},\mathcal{Q})$, quality requirements given by objective functions $\{f_i\}_{i\in I}$ and constraints $\{c_j\}_{j\in J}$, and designer-specified tolerances $\{\gamma_k\}_{k\in K}$ for the continuous parameters of the system. Also, let $\mathcal{F}$ be the set of feasible designs for the system (i.e., of candidate designs that meet the tolerances $\{\gamma_k\}_{k\in K}$ and satisfy the constraints $\{c_j\}_{j\in J}$):

$$\begin{aligned}
\mathcal{F} = \big\{\mathcal{C}(\mathcal{P}',q) \;\big|\; \mathcal{P}' = \mathsf{X}_{k\in K}\left[k'^{\perp}, k'^{\top}\right] \subset \mathcal{P} \wedge q\in\mathcal{Q} \wedge\\
\forall k\in K.\, k'^{\top}-k'^{\perp} = 2\gamma_k(k^{\top}-k^{\perp}) \wedge \forall j\in J.\, c_j(\mathcal{C}(\mathcal{P}',q))\big\}.
\end{aligned} \quad (11)$$

Definition 4. *A* sensitivity-aware Pareto dominance relation *over a feasible design set $\mathcal{F}$ and a set of minimisation objective functions $\{f_i\}_{i\in I}$ is a relation $\prec \subset \mathcal{F}\times\mathcal{F}$ such that for any feasible designs $d, d'\in\mathcal{F}$*

$$\begin{aligned}
d\prec d' \iff\\
&\big(\forall i\in I.\, f_i(d)\le f_i(d') \wedge\ \exists i\in I.\,(1+\epsilon_i)f_i(d) < f_i(d')\big) \vee\\
&\big(\forall i\in I.\, f_i(d)\le f_i(d') \wedge\ \exists i\in I.\, f_i(d) < f_i(d') \wedge\\
&\qquad sens(d)\le sens(d')\big).
\end{aligned} \quad (12)$$

where the objective functions $\{f_i\}_{i\in I}$ are calculated using one of the alternative definitions from Table 1 and $\epsilon_i \ge 0$ are sensitivity-awareness parameters.

The *parametric Markov chain synthesis problem* consists of finding the Pareto-optimal set PS of candidate designs (5) (i.e. pCTMCs) with tolerances $\{\gamma_k\}_{k\in K}$ that satisfy the constraints $\{c_j\}_{j\in J}$ and are *non-dominated* with respect to the objective functions $\{f_i\}_{i\in I}$ and the sensitivity-aware dominance relation '$\prec$':

$$PS = \left\{\mathcal{C}(\mathcal{P}',q)\in\mathcal{F} \;\middle|\; \nexists \mathcal{C}(\mathcal{P}'',q')\in\mathcal{F}.\ \mathcal{C}(\mathcal{P}'',q')\prec\mathcal{C}(\mathcal{P}',q)\right\}, \quad (13)$$

Before discussing the rationale for this definition, we show that the sensitivity-aware Pareto dominance relation is a strict order like the classical Pareto dominance.

Theorem 1. *The sensitivity-aware Pareto dominance relation is a strict order.*

Proof. See Appendix A □

The classical Pareto dominance definition can be obtained by setting $\epsilon_i = 0$ for all $i \in I$ in (12). When $\epsilon_i > 0$ for some $i \in I$, dominance with respect to quality attribute i holds in our generalised definition in two scenarios:

1) when the quality attribute has a much lower value for the dominating design, i.e. $(1+\epsilon_i) f_i(d) < f_i(d')$;
2) when in addition to a (slightly) lower quality attribute value, i.e. $f_i(d) < f_i(d')$, the sensitivity of the dominating design is no worse than that of the dominated design, i.e. $sens(d) \leq sens(d')$.

These scenarios are better aligned with the needs of designers than those obtained by using sensitivity as an additional optimisation criterion, which induces Pareto fronts comprising many designs with low sensitivity but unsuitably poor quality attributes. Similarly, each objective function definition from Table 1 captures specific needs of real-world systems. Thus, using the "upper bound" definition ($f_i^{\top}$) in (12) supports the synthesis of *conservative designs* by comparing competing designs based on the worst-case values of their quality attributes. This is suitable when the worst-case performance, reliability, etc. must be specified for a system, e.g. in its service-level agreement. In contrast, the "lower bound" definition from Table 1 ($f_i^{\perp}$) can be used when design selection must be based on the best expected quality values of a system. Finally, the "mid-range" definition ($f_i^{\bullet}$) may be useful—in conjunction with the actual sensitivity (10)—to compare and select designs based on their reference midpoint quality values.

Importantly, for $\epsilon_i > 0$ our generalised definition induces Pareto fronts comprising designs with non-optimal (in the classical sense) objective function values, but with low sensitivity. We call such designs *sub-optimal robust*. Thus, ϵ_i can be finely tuned to sacrifice objective function optimality (slightly) for better robustness. Below we formally characterize the set of robust sub-optimal designs and provide an example of the sensitivity-aware dominance relation.

Definition 5 (Sub-optimal robust design)**.** *Let PS be a Pareto-optimal set defined as per (13). A design $d' \in PS$ is called* robust sub-optimal *if $\exists d \in PS$ s.t.:*

$$\big(\forall i \in I.\, f_i(d) \leq f_i(d') \wedge \exists i \in I.\, f_i(d) < f_i(d')\big)$$

Example 5 (Sensitivity-aware Pareto dominance relation)**.** *Consider the quality-attribute regions of Figure 4 induced by designs d_1, d_2, d_3 of the producer-consumer model introduced in Examples 1-4, and the objective functions defined as $f_i = f_i^{\perp}$ for $i \in 1, 2$. Visually, $f_i^{\perp}$ corresponds to the lower-left corners of the regions in Figure 4. Since we maximize both objectives, for clarity, we report below the dominance relation for maximisation:*

$$\begin{aligned} d \succ d' \iff & \\ & \big(\forall i \in I.\, f_i(d) \geq f_i(d') \wedge\ \exists i \in I.\, f_i(d) > (1+\epsilon_i) f_i(d')\big) \vee \\ & \big(\forall i \in I.\, f_i(d) \geq f_i(d') \wedge\ \exists i \in I.\, f_i(d) > f_i(d') \wedge \\ & \qquad sens(d) \leq sens(d')\big). \end{aligned}$$

The designs d_1, d_2, d_3 have identical parameter tolerances and thus, same parameter space volume V. We have that $d_1 \succ d_2 \succ d_3$ when $\epsilon_1 = \epsilon_2 = 0$ (classical dominance) because for $i = 1, 2$, $f_i^{\perp}(d_1) > f_i^{\perp}(d_2), f_i^{\perp}(d_3)$. Further, we have that $d_1 \not\succ d_2$ when $\epsilon_1 = \epsilon_2 = 0.05$, implying that d_2 is robust sub-optimal, i.e., is retained in the sensitivity-aware Pareto-optimal set, because $f_1^{\perp}(d_1) \not> 1.05 \cdot f_1^{\top}(d_2)$, $f_2^{\perp}(d_1) \not> 1.05 \cdot f_2^{\perp}(d_2)$, and $sens(d_1) \not\leq sens(d_2)$. Design d_3 is not included in the front ($d_1, d_2 \succ d_3$) because $f_i^{\perp}(d_1), f_i^{\perp}(d_2) > 1.05 \cdot f_i^{\perp}(d_3)$ for $i = 1, 2$.

4. Synthesis of Sensitivity-Aware Pareto Sets

In this section, we describe our method for computing sensitivity-aware Pareto sets. The method employs genetic multi-objective optimisation algorithms for generating candidate designs and a precise parameter analysis of pCTMCs for evaluating the candidate designs. We start with a method overview, then we describe the two components the method builds on.

4.1. Method Overview

Computing the Pareto-optimal design set (13) using exhaustive analysis is very expensive and requires a significant amount of computational resources as the design space $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ is extremely large due to its real-valued parameters. Also, every candidate design $\mathcal{C}(\mathcal{P}', q)$ consists of an infinite set of CTMCs that cannot all be analysed to establish its quality and sensitivity. To address these challenges, our pCTMC synthesis method combines search-based software engineering (SBSE) techniques [24] with techniques for effective pCTMCs analysis [23, 25], producing a close approximation of the Pareto-optimal design set.

Algorithm 1 presents the high-level steps of our pCTMC synthesis method. The approximate Pareto-optimal design set $\overline{PS}$ returned by this algorithm starts empty (line 2) and is assembled iteratively by the while loop in lines 3–16 until a termination criterion $\textsc{Terminate}(\mathcal{C}(\mathcal{P}, \mathcal{Q}), \overline{PS})$ is satisfied. Each iteration of this while loop uses an SBSE metaheuristic to get a new set of candidate designs (line 4) and then updates the approximate Pareto-optimal design set $\overline{PS}$ in the for loop from lines 5–15. This update involves analysing each candidate design $d = \mathcal{C}(\mathcal{P}', q)$ to establish its associated objective function and constraint values in line 6, where we use the shorthand notation $f_{i,d}^{\top} \equiv f_i^{\top}(\mathcal{C}(\mathcal{P}', q))$, $f_{i,d}^{\perp} \equiv f_i^{\perp}(\mathcal{C}(\mathcal{P}', q))$ and $c_{j,d} \equiv \forall p \in \mathcal{P}'.c_j(\mathcal{C}(p, q))$ for all $i \in I$, $j \in J$. If the design satisfies all

Algorithm 1 Parametric Markov chain synthesis

```
1: function SYNTHESIS(C(P,Q), {f_i}_{i∈I}, {c_j}_{j∈J}, {γ_k}_{k∈K})
2:     PS ← ∅
3:     while ¬TERMINATE(C(P,Q), PS) do
4:         CD ← CANDIDATEDESIGNS(C(P,Q), {γ_k}_{k∈K}, PS)
5:         for all d ∈ CD do
6:             ({f_{i,d}^⊤}_{i∈I}, {f_{i,d}^⊥}_{i∈I}, {c_{j,d}}_{j∈J}) ←
                   ANALYSEDESIGN(d, {f_i}_{i∈I}, {c_j}_{j∈J})
7:             if ⋀_{j∈J} c_{j,d} then
8:                 dominated = false
9:                 for all d' ∈ PS do
10:                    if d' ≺ d then dominated = true; break
11:                    if d ≺ d' then PS = PS \ {d'}
12:                end for
13:                if ¬dominated then PS = PS ∪ {d}
14:            end if
15:        end for
16:    end while
17:    return PS
18: end function
```

constraints (line 7), the for loop in lines 9-12 finds out if the new design d is dominated by, or dominates, any designs already in $\overline{PS}$. Existing designs dominated by d are removed from $\overline{PS}$ (line 11), and d is added to the Pareto-optimal design set if it is not dominated by any existing designs (line 13).

The elements below must be concretised in the synthesis algorithm, and are described in the next two sections:

1) The AnalyseDesign function for establishing the quality attributes and constraint compliance of a candidate design;
2) The CandidateDesigns SBSE metaheuristic and the associated Terminate criterion.

The time complexity of Algorithm 1 is *linear* with respect to the overall number of optimisation objectives and constraints and the time required to analyse one quality attribute of a candidate design. The complexity is further affected by the SBSE metaheuristic setting, namely by the number of generations k (i.e. the number of iterations of the while loop) and the size of the candidate design population $N{=}|CD|$. Increasing the total number of design evaluations (i.e. $k \cdot N$) typically improves the Pareto optimality of the generated design set, but also slows down the synthesis process. We provide a detailed complexity analysis of the synthesis process in Appendix B.

4.2. Computing Safe Property Bounds for pCTMCs

To establish the quality attributes and sensitivity of candidate designs, AnalyseDesign uses precise parameter synthesis techniques [23] to compute safe enclosures of the satisfaction probability of CSL formulae over pCTMCs. Given a pCTMC $\mathcal{C}(\mathcal{P}', q)$ and a CSL path formula ϕ, these techniques provide a safe under-approximation $\Lambda^q_{\min}$ and a safe over-approximation $\Lambda^q_{\max}$ of the minimal and maximal probability that $\mathcal{C}(\mathcal{P}', q)$ satisfies ϕ:

$$\Lambda^q_{\min} \leq \inf_{p \in \mathcal{P}'} \Lambda_\phi(p, q) \;\text{ and }\; \Lambda^q_{\max} \geq \sup_{p \in \mathcal{P}'} \Lambda_\phi(p, q).$$

This supports the safe approximation of the bounds $\{f_i^\perp, f_i^\top\}_{i \in I}$ of the objective functions and of the constraints $\{c_j\}_{j \in J}$. As shown in [23], the over-approximation quality improves as the size of $\mathcal{P}'$ decreases. Therefore, the precision of the approximation can be effectively controlled via parameter space decomposition, where $\mathcal{P}'$ is decomposed into subspaces $\mathcal{P}'_1$, $\mathcal{P}'_2 \ldots \mathcal{P}'_n$ and $\Lambda^q_{\min}$ ($\Lambda^q_{\max}$) is taken as the minimum (maximum) of the bounds computed for these n subspaces. Although this refinement step improves the precision of bounds, it also increases the complexity of AnalyseDesign n-fold [23].

The satisfaction function Λ_ϕ is typically non-monotonic (and, for nested properties, non-continuous), so safe bounds cannot be obtained by simply evaluating Λ_ϕ at the extrema of parameter region $\mathcal{P}'$. Accordingly, our technique builds on a parametric backward transient analysis that computes safe bounds for the parametric transient probabilities in the discrete-time process derived from the pCTMC. This discretisation is obtained through standard uniformisation, and through using the Fox and Glynn algorithm [22] to derive the required number of discrete steps for a given time bound. Once the parametric discrete-time process is obtained, the computation of the bounds reduces to a local and stepwise minimisation/maximisation of state probabilities in a time non-homogenous Markov process. Presenting the technique in detail as well as the analysis of the approximation error is outside the scope of our paper, but the interested reader can find a complete description in [23].

Our approach can be easily extended to also support time-unbounded properties by using the method of [26] for parameter synthesis of discrete-time Markov models and properties expressed by time-unbounded formulae of probabilistic computation tree logic.

4.3. Metaheuristic for Parametric CTMC Synthesis

To ensure that CandidateDesigns selects suitable candidate designs, Algorithm 1 is implemented as a *multiobjective optimisation genetic algorithm* (MOGA) such as NSGA-II [27] or MOCell [28]. MOGAs are genetic algorithms specifically tailored for the synthesis of close Pareto-optimal set approximations that are spread uniformly across the search space. As with any genetic algorithm [29], possible solutions—candidate designs in our case—are encoded as tuples of *genes*, i.e. values for the problem variables. In particular, any candidate design $\mathcal{C}(\mathcal{P}', q)$ that satisfies a fixed set of tolerances $\{\gamma_k\}_{k \in K}$ is uniquely encoded by the gene tuple (p, q), where $p \in \mathcal{P}$ is the centre point of the continuous parameter region $\mathcal{P}'$. The structure of the gene tuple (p, q) for any pCTMC $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ is automatically extracted through parsing the evolvable constructs (4). This feature enables to conveniently encode the pCTMC parameters into a representation suitable for the MOGAs.

Consider the candidate designs d_1, d_2, d_3 with tolerance value $\gamma = 0.005$ from Example 2. The gene tuple (p, q) of a candidate design $\mathcal{C}(\mathcal{P}', q)$ has the structure (r_slow_rate, delta_rate, module_idx)*, where* module_idx $\in \{1, 2\}$ *is the index of the Buffer module used by the candidate design. Thus, the designs d_1, d_2, d_3 have gene tuples given by* $(15.145, 2.08, 2)$*,* $(13.325, 3.66, 2)$ *and* $(17.365, 2.93, 1)$*, respectively.*

The first execution of CandidateDesigns from Algorithm 1 returns a randomly generated *population* (i.e. set) of feasible designs (11). This population is then iteratively evolved by subsequent CandidateDesigns executions into populations of "fitter" designs through MOGA *selection*, *crossover* and *mutation*. Selection chooses the population for the next iteration and a *mating pool* of designs for the current iteration by using the objective functions $\{f_i\}_{i \in I}$, the sensitivity-aware dominance relation (12) and the distance in the parameter space $\mathcal{P}$ between designs to evaluate each design. Crossover randomly selects two designs from the mating pool, and generates a new design by combining their genes, and mutation yields a new design by randomly modifying some of the genes of a design from the pool.

The evolution of the design population terminates (i.e. the predicate Terminate$(\mathcal{C}(\mathcal{P}, \mathcal{Q}), \overline{PS})$ returns true) after a fixed number of design evaluations or when a predetermined number of successive iterations generate populations with no significantly fitter designs.

The implementation of the selection, crossover and mutation operations is specific to each MOGA. For instance, [27] presents these features for the NSGA-II MOGA used in our experimental evaluation from Section 6.

5. RODES: A Robust-Design Synthesis Tool

Our GPU-accelerated RODES tool synthesises sensitivity-aware Pareto sets by implementing the process described in Algorithm 1. In this section, we first present the architecture of RODES, and then describe how we achieved significant performance and scalability improvements through the use of a two-level parallelisation for the synthesis process.

5.1. RODES Architecture

As shown in Figure 5, the operation of RODES is managed by a *Robust-design synthesis engine*. First, a *Model parser* (built using the Antlr parser generator, `www.antlr.org`) preprocesses the design-space pCTMC model. Next, a *Sensitivity-aware synthesiser* uses the jMetal Java framework for multi-objective optimisation with metaheuristics (`jmetal.github.io/jMetal`) to evolve an initially random population of *candidate designs*, generating a close approximation of the sensitivity-aware Pareto front. This involves using a *Candidate design analyser*, which invokes the probabilistic model checker PRISM-PSY [25] to obtain the ranges of values for the relevant quality attributes of candidate designs through precise parameter synthesis. The Pareto front and corresponding Pareto-optimal set of designs are then plotted using MATLAB/Octave scripts, as shown in Figure 7.

A key feature of RODES is its modular architecture. The Sensitivity-aware synthesiser supports several metaheuristics algorithms, including variants of genetic algorithms and swarm optimisers. Furthermore, the sensitivity-aware Pareto dominance relation can be adapted to match better the needs of the system under development (e.g., by comparing designs based on the worst, best or average quality attribute values). Finally, different solvers could be used for the probabilistic model checker component, including the parameter synthesis solvers for discrete-time Markov chains and time unbounded properties [26] implemented in the tools PROPhESY [30] and STORM [31].

The open-source code of RODES is available on our project website `https://github.com/gerasimou/RODES`.

5.2. Two-Level Parallelisation

Synthesising sensitivity-aware Pareto sets is a computationally expensive process. To mitigate the performance issues that could arise due to the increased total number of evaluations $(k \cdot N)$ or the complexity of evaluating candidate designs (t), we employ a two-level parallelisation.

At the first level, we exploit the fact that the evaluations of particular candidates within a single population are independent and thus they can run in parallel (line 6 in Algorithm 1). A synchronisation is required only after all candidates are evaluated to update the approximate Pareto-optimal set $\overline{PS}$ and to generate new candidates. This granularity of parallelism allows us to efficiently utilise both multi-core and multi-processor architectures. In particular, we can span in parallel a number of tasks that is equal to the population size N and thus significantly alleviate the complexity corresponding to the total number of design evaluations per MOGA generation. We can further increase the parallelisation at this level given that the evaluation of quality attributes for each design is independent. Thus, we can span up to $N \cdot (|I| + |J|)$ tasks to evaluate these attributes in parallel and reduce the computation time. The current RODES implementation supports parallelisation at the population level but not at the level of quality attributes, which we plan to add in future tool releases.

The second level of parallelisation aims at accelerating the evaluation of a single candidate over a single quality attribute. The key factor affecting the time t required to analyse a quality attribute of a candidate design is the size of the candidate, namely, the number of non-zero elements M in the rate matrix of the underlying pCTMC. This number is proportional to the number of states in the pCTMC, and reflects the complexity of the candidate designs. To ensure that RODES supports robust design synthesis for complex systems comprising up to tens thousands of states, our second-level parallelisation improves scalability with respect to the number of states. In par-

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

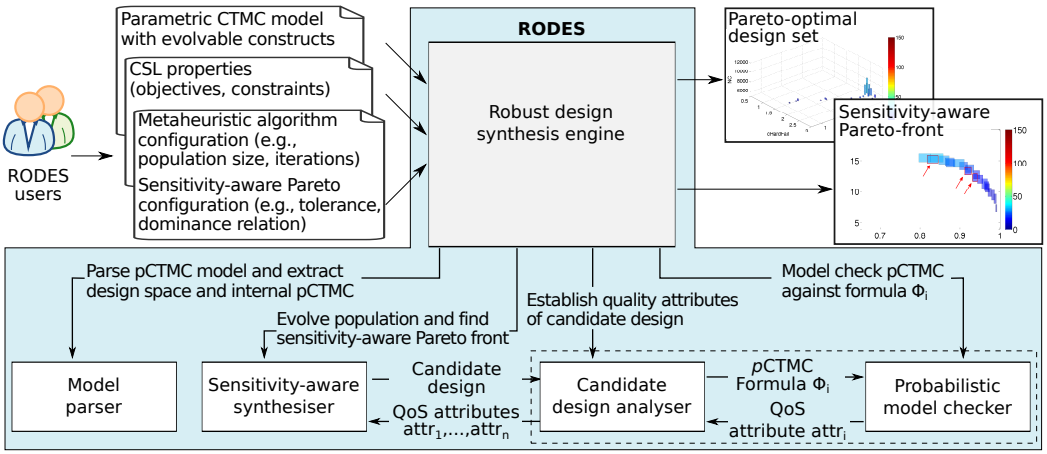


Figure 5: High-level RODES architecture.

ticular, we build on our previous work [25] to integrate a GPU acceleration of the *p*CTMC analysis into RODES.

This parallelisation is much more involved, since the computation for individual states is not independent. As such, the *p*CTMC analysis is formulated in terms of matrix-vector operations, making it suitable for effective data-parallel processing. Accordingly, RODES implements a state space parallelisation, where a single row of the parametric rate matrix (corresponding to the processing of a single state) is mapped to a single computational element. As the underlying *p*CTMCs typically have a balanced distribution of the state successors, this mapping yields a balanced distribution of non-zero elements in the rows of the matrix. The outcome is a good load balancing within the computation elements, leading to significant acceleration. In contrast to the parallelisation proposed in [25], RODES is designed to leverage the computational power of modern GPUs, which provide hundreds of computational elements and can schedule thousands of active threads in a different way. In particular, RODES can evaluate on a single GPU several candidate designs (that can differ both in their discrete and in their continuous parameters) in parallel, provided that the underlying *p*CTMCs can fit in the GPU memory. This enables an efficient and flexible utilisation of the available computation power for complex robust design synthesis problems (see performance evaluation results in Section 6.4).

6. Evaluation

We evaluate the effectiveness of RODES using three systems from different application domains. Also, we assess the performance and scalability of RODES including the impact of the two-level parallelisation. We conclude our evaluation with a discussion of threats to validity.

6.1. Research Questions

The aim of our experimental evaluation was to answer the following research questions.

RQ1 (Decision Support): Can RODES support decision making by identifying effective tradeoffs between the QoS optimality and the sensitivity of alternative designs? To support decision making, RODES must provide useful insights into the robustness of alternative system designs. Therefore, we assessed the optimality-sensitivity tradeoffs suggested by RODES for the software systems used in our evaluation.

RQ2 (Performance): Does the two-level paralellisation improve the efficiency of RODES? Since the synthesis of robust models is a computationally expensive process, we examined the change in performance thanks to the two-level parallelisation architecture described in Section 5.2.

RQ3 (Metaheuristic Effectiveness): How does our RODES approach perform compared to random search? Following the standard practice in search-based software engineering [32], we assessed if the stochastic models synthesised by RODES "comfortably outperform" those synthesised by a random search approach.

6.2. Analysed Software Systems

We performed a wide range of experiments to evaluate our RODES approach and tool using three software systems from different application domains:

- a producer-consumer (PC) software system described in Examples 1-5;
- a replicated file system used by Google's search engine [33];
- a cluster availability management system [34].

We have already presented the PC system in Examples 1-5. In the following paragraphs, we introduce the other systems, provide a description of their stochastic models and present the objectives and constraints used

to synthesise robust Pareto optimal designs. Further information about these systems are available on our project website at `https://github.com/gerasimou/RODES/wiki`.

Google File System (GFS). GFS partitions files into chunks of equal size, and stores copies of each chunk on multiple *chunk servers*. A master server monitors the locations of these copies and the chunk servers, replicating the chunks as needed. During normal operation, GFS stores CMAX copies of each chunk. However, as servers fail and are repaired, the number c of copies for a chunk may vary from 0 to CMAX.

Previous work modelled GFS as a CTMC with fixed parameters and focused on the analysis of its ability to recover from disturbances (e.g. c<CMAX) or disasters (e.g. master server down) [33]. In our work, we adapt the CTMC of the lifecycle of a GFS chunk from [33] by considering several continuous and discrete parameters that a designer of the system has to decide. Figure 6 shows the resulting model, encoded in the PRISM modelling language extended with the evolve constructs from (4). As in [33], we model separately the software and hardware failures and repairs, for both the master server (lines 22–25) and the chunk servers (lines 26–31), and assume that loss of chunk copies due to chunk server failures leads to further chunk replications, which is an order of magnitude slower if $c = 0$ and a backup of the chunk must be used (line 32).

To evaluate RODES, we assume that GFS designers must select the hardware failure and repair rates cHardFail and cHardRepair of the chunk servers, and the maximum number of chunks NC stored on a chunk server within the ranges indicated in Figure 6. These parameters reflect the fact that designers can choose from a range of physical servers, can select different levels of service offered by a hardware repair workshop, and can decide a maximum workload for chunk servers. We consider an initial system state modelling a severe hardware disaster with all servers down due to hardware failures and all chunk copies lost, and we formulate a pCTMC synthesis problem for quality requirements given by two maximising objective functions and one constraint:

f_1: $P_{=?}\,[\neg \mathsf{SL1}\; U^{[10,60]}\; \mathsf{SL1}]$, where $\mathsf{SL1} = \mathsf{M\_up} \wedge \mathsf{c} > 0$ holds in states where *service level 1* (master up and at least one chunk copy available) is provided;

f_2: $R\{\text{“}active\text{”}\}_{=?}\;[C^{<=60}]$, where a reward of 1 is assigned to the states with a number of running chunk servers of at least 0.5M (i.e., half of the total number of chunk servers);

c_1: $R\{\text{“}replicates\text{”}\}_{\leq 5}\;[C^{<=60}]$, where a transition reward of 1 is assigned to each chunk replication transition.

Objective f_1 maximises the probability that the system recovers service level 1 in the time interval $[10, 60]$ hours. Objective f_2 maximises the expected time the system stays in (optimal) states with at least $0.5M$ chunk servers up in the first 60 hours of operation. Finally, constraint c_1 restricts the number of expected chunk replications over 60 hours of operations.

```
1  ctmc
   // Failure rates
2  const double mSoftFail = 0.000475;     // master software
3  const double mHardFail = 0.000025;     // master hardware
6  const double cSoftFail = 0.475;        // chunk server software
7  evolve double cHardFail [0.25..4.0];   // chunk server hardware
   // Repair rates
4  const double mSoftRepair = 12;         // master software
5  const double mHardRepair = 6;          // master hardware
8  const double cSoftRepair = 12;         // chunk server software
9  evolve double cHardRepair [0.5..4.0];  // chunk server hardware
10 const int N=100000;                    // total number of GFS chunks
11 const int M=20;                        // number of chunk servers
12 evolve int NC [5000..20000];           // max chunks per chunk server
13 const int CMAX=3;                      // optimal number of chunk copies

14 module GFS_Chunk
15  M_up : bool init false;      // master is up
16  M_sdown : bool init false;   // master is down with SW problem
17  M_hdown : bool init true;    // master is down with HW problem
18  Cup : [0..M] init 0;         // number of chunk servers up
19  Csdown : [0..M] init 0;      // number of chunk servers down (SW problem)
20  Chdown : [0..M] init 20;     // number of chunk servers down (HW problem)
21  c : [0..CMAX] init 0;        // number of chunk copies available
   // Master server failure and repair
22  [] M_up → mSoftFail : (M_up'=false)&(M_sdown'=true);
23  [] M_up → mHardFail : (M_up'=false)&(M_hdown'=true);
24  [] M_sdown → mSoftRepair : (M_up'=true)&(M_sdown'=false);
25  [] M_hdown → mHardRepair : (M_up'=true)&(M_hdown'=false);
   // Chunk servers failure and repair
26  [] Cup>0 & c>0 & Csdown<M → (c/Cup)*cSoftFail :
                    (Cup'=Cup-1)&(Csdown'=Csdown+1)&(c'=c-1);
27  [] Cup>0 & Cup>c & Csdown<M → (1-(c/Cup))*cSoftFail :
                    (Cup'=Cup-1)&(Csdown'=Csdown+1);
28  [] Cup>0 & c>0 & Chdown<M → (c/Cup)*cHardFail :
                    (Cup'=Cup-1)&(Chdown'=Chdown+1)&(c'=c-1);
29  [] Cup>0 & Cup>c & Chdown<M → (1-(c/Cup))*cHardFail :
                    (Cup'=Cup-1)&(Chdown'=Chdown+1);
30  [] Cup<M & Csdown>0 → Csdown*cSoftRepair :
                    (Csdown'=Csdown-1)&(Cup'=Cup+1);
31  [] Cup<M & Chdown>0 → cHardRepair :
                    (Chdown'=Chdown-1)&(Cup'=Cup+1);
32  [] M_up & c<CMAX & Cup>c & Cup*NC>=(c+1)*N →
                                   ((c>0)?20:2):(c'=c+1);
33 endmodule
```

Figure 6: pCTMC model of the Google File System

Workstation Cluster (WC). We extend the CTMC of a cluster availability management system from [34]. This CTMC models a system comprising two sub-clusters, each with N workstations and a switch that connects the workstations to a central backbone. For each component, we consider failure, inspection and repair rates (where repairs are initiated only after an inspection detects failures), and we assume that designers must decide these rates for workstations—i.e., the real-valued parameters wsFail, wsCheck and wsRepair for our pCTMC, respectively. Additionally, we assume that designers must select the sub-cluster size N, and must choose between an expensive repair implementation (i.e., pCTMC module) with a 100% success probability and a cheaper repair module with 50% success probability—i.e., two discrete parameters for the pCTMC. We made this model available on our repository of case studies.

For an initial system state with 5 workstations active in each sub-cluster and switches and backbone working, we formulate a pCTMC synthesis problem for quality requirements given by two maximising objective functions

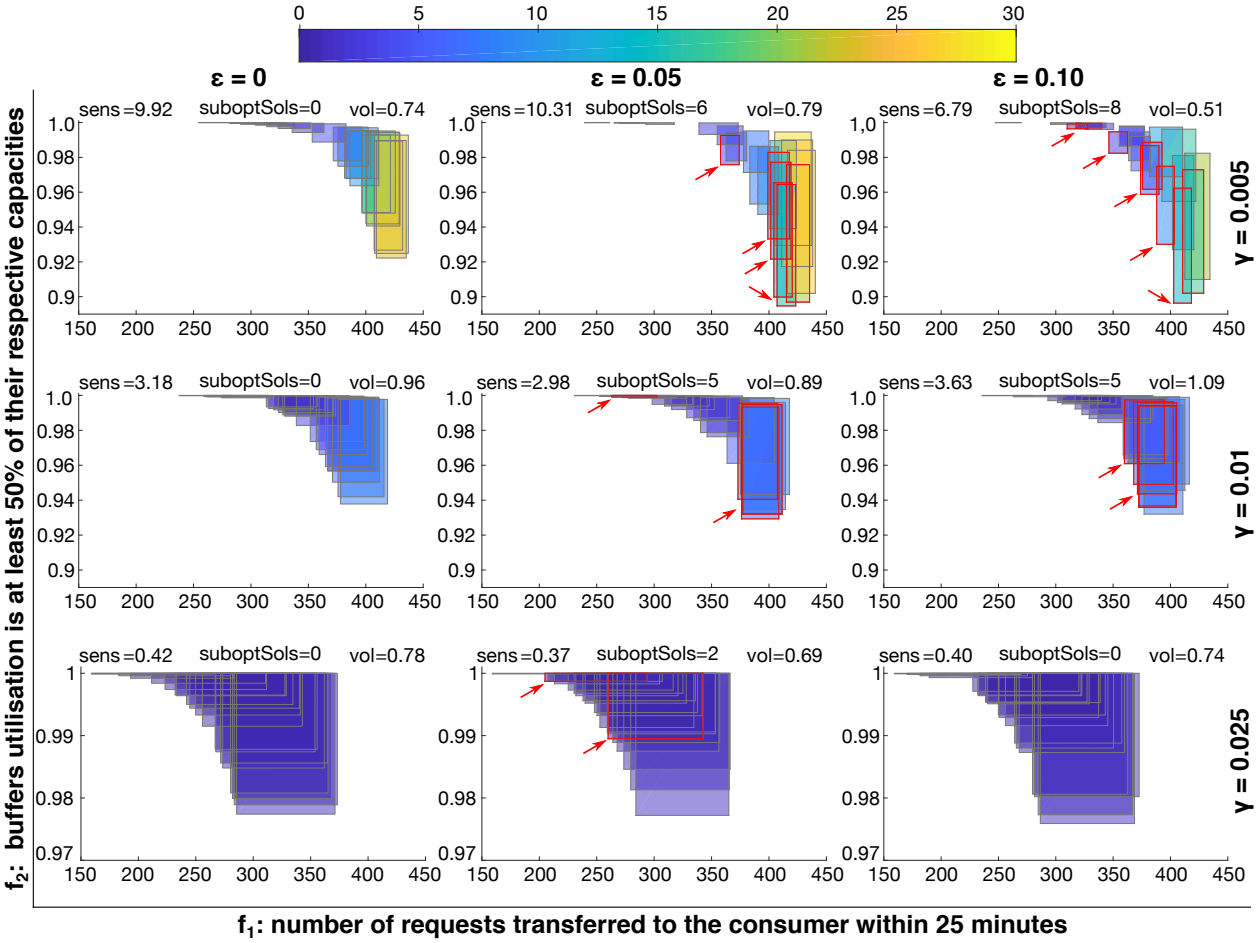


Figure 7: Sensitivity-aware Pareto fronts for the producer-consumer model. Boxes represent quality-attribute regions, coloured by sensitivity (yellow: sensitive, blue: robust). Red-bordered boxes indicate sub-optimal robust designs. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box). Statistics are: sens, average sensitivity of the front; suboptSols, number of suboptimal solutions; vol, average volume of the front.

and one constraint:

f_1: $P_{=?}$ [¬premium $U[20,100]$ premium], where premium denotes a system service where at least $1.25N$ workstations are connected and operating;

f_2: $R\{\text{“}operational\text{”}\}_{=?}[C^{\leq 100}]$, where a reward of 1 is assigned to states with a number of operating clusters between $1.2N$ and $1.6N$;

c_1: $R\{\text{“}repair\text{”}\}_{\leq 80}$ $[C^{\leq 100}]$, where transition rewards are associated with repair actions of the workstations, backbone and switches.

Objective f_1 maximises the probability that the system recovers the premium service in the time interval $[20, 100]$ hours. Objective f_2 maximises the expected time the system spends in cost-optimal states during the first 100 hours of operation. Constraint c_1 restricts the cost of repair actions during this time (the definition of the cost is provided on our project website).

6.3. Evaluation methodology

We used the following configuration to evaluate RODES: NSGA-II MOGA, 10000 evaluations, initial population of 20 individuals, and default values for single-point crossover probability $p_c = 0.9$ and single-point mutation probability $p_m = 1/(|K| + |D|)$, with $|K| + |D|$ the number of (continuous and discrete) design-space parameters. We examine the behaviour of the sensitivity-aware Pareto dominance relation using different combinations of tolerance values $\gamma \in \{0.005, 0.01, 0.025\}$ and sensitivity-awareness coefficients $\epsilon_i \in \{0.00, 0.05, 0.10\}$.

For each experiment, we report the sensitivity-aware Pareto fronts (Figures 7, 9, 12 and 14). The Pareto-optimal designs are depicted as boxes in the quality-attribute space and coloured by sensitivity, using the same representation as in Figures 1 and 4. We also show the synthesised designs in the design space, given by the continuous and discrete parameters of the system. In this case, designs are represented as boxes in the continuous parameter space, representing the extent of the parameter variation under the given tolerance. The third dimension (vertical axis) in Figures 10 and 13 gives the value of the discrete parameter.

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

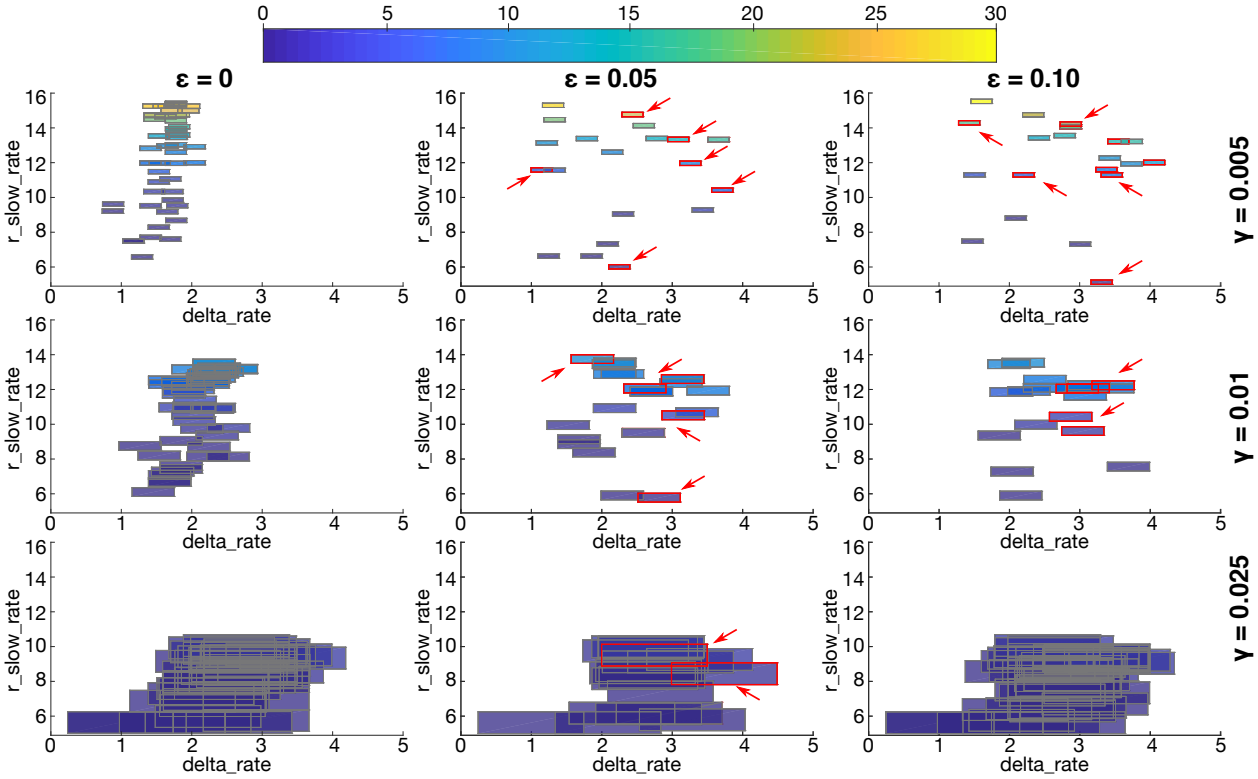


Figure 8: Synthesised Pareto-optimal designs for the producer-consumer model and experiments from Figure. 7. Rectangles in x-y plane correspond to the continuous parameter regions. The discrete parameter (module - 'mod') is omitted since RODES synthesised solutions using only the redirection module ('mod2'). Boxes are coloured by sensitivity.

6.4. Results and Discussion

RQ1 (Decision Support). We analysed the designs synthesised by RODES in order to identify actionable insights regarding the tradeoffs between the QoS attributes and sensitivity of alternative architecture designs. For each system, we present our findings independently.

Producer-consumer system (PC). First, we present the results for the producer consumer system introduced in Examples 1-5, obtained by running our RODES tool with tolerances $\gamma \in \{0.005, 0.01, 0.025\}$ for both continuous parameters (r_slow_rate and delta_rate). The resulting Pareto fronts are shown in Figure 7, for objectives f_1 (number of requests transferred to the consumer within 25 minutes) and f_2 (probability of adequate buffer utilization) and sensitivity-awareness parameters $\epsilon_1 = \epsilon_2 = \epsilon \in \{0, 0.05, 0.1\}$. The corresponding synthesised designs are presented in Figure 8.

These Pareto fronts provide a wealth of information supporting the evaluation of the optimality and robustness of alternative designs. In particular, the Pareto front for $\epsilon = 0$ and $\gamma = 0.005$ contains several large (yellow) boxes that correspond to highly sensitive designs. Increasing ϵ produces a number of robust sub-optimal designs (red-bordered) with slightly sub-optimal quality attributes but improved robustness. Such designs represent valuable alternatives to the highly sensitive solutions obtained using the classical, sensitivity-agnostic, dominance relation. This ability to identify poor (i.e. highly sensitive) designs and then alternative robust designs with similar quality attributes is a key and unique benefit of our design synthesis method. Consider for instance the results for $\epsilon = 0.05$ and $\gamma = 0.005$. There are several sensitive designs at high f_1 values (see Figure 7), which correspond to designs with r_slow_rate above 15 and low values delta_rate (below 2.5), see Figure 8. Through our method, we found that there exist alternative sub-optimal designs with improved robustness (highlighted green boxes), corresponding to higher delta_rate and lower r_slow_rate values, i.e, to designs with a slower slow buffer and a faster fast buffer.

Furthermore, we observe that the overall sensitivity improves as the tolerance γ increases, meaning that the uncertainty (volume) of the quality attribute regions grows proportionally smaller than the uncertainty of the corresponding parameter regions, see (10). This explains why we observe fewer sub-optimal robust designs for higher tolerances ($\gamma = 0.01, 0.025$). Increasing parameter tolerances also affects the quality attribute profiles as it leads to larger ranges for objective f_1 (i.e., more sensitive) and to smaller ranges for f_2 (i.e., more robust). As a consequence, RODES tends to favour Pareto-optimal solutions with better f_2 and worse f_1 values as the tolerance increases. In particular, for $\gamma = 0.025$ all designs with $f_1^{\perp} \geq 300$ are excluded (corresponding to the most sensitive designs for

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

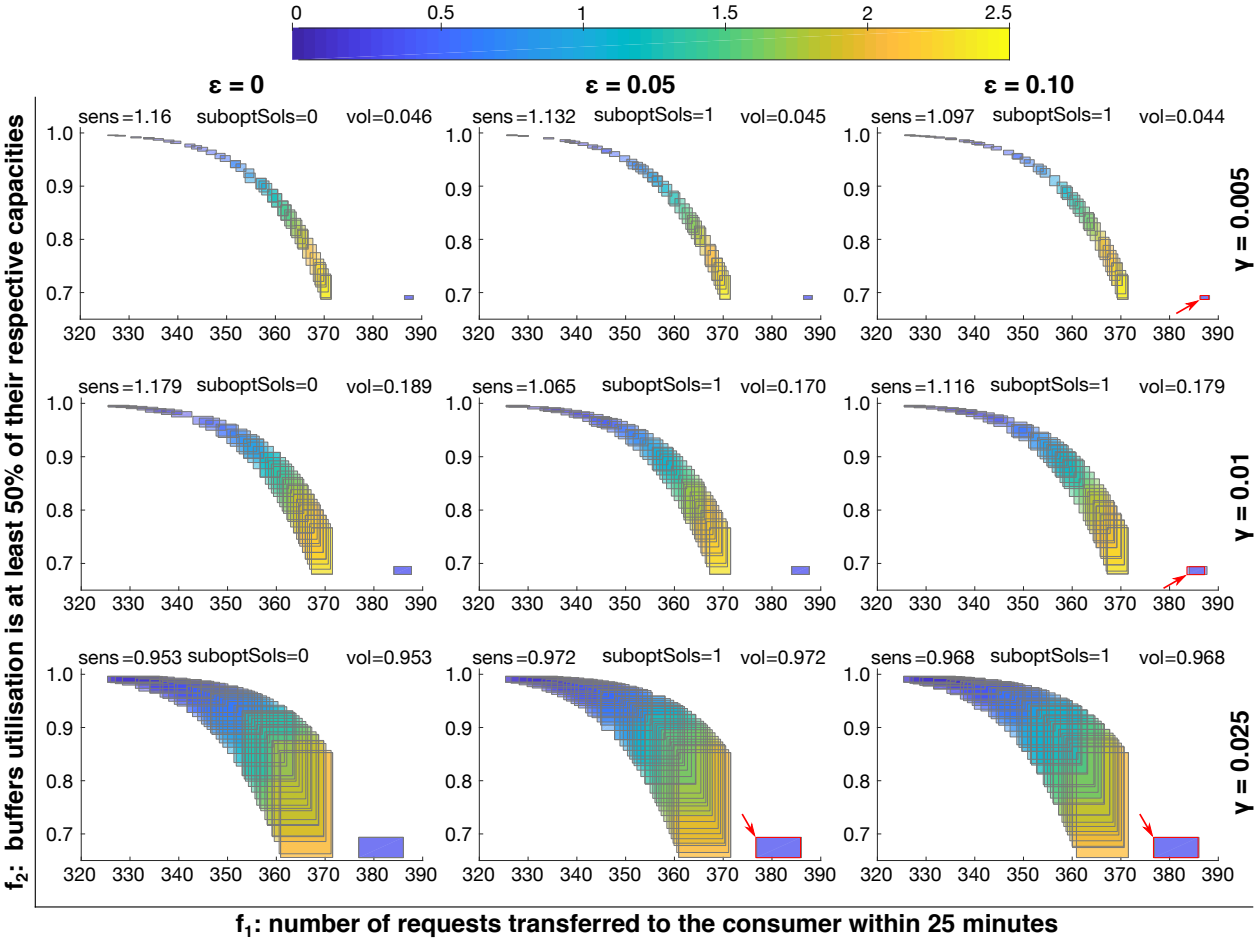


Figure 9: Sensitivity-aware Pareto fronts for the second variant of the producer-consumer model. Legend and colour code are as in Figure 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

$\gamma = 0.005, 0.01$), which yields regions with average volume comparable to those for $\gamma = 0.025$.

The synthesised parameter regions (Figure 8) indicate that redirection (second module – 'mod2') is always preferred to non-redirection. Also, the generated designs select values for the continuous parameters from the lower-end of their respective range, with r_slow_rate $\in [5.00, 15.650]$ and delta_rate $\in [0.242, 4.489]$. In other words, our algorithm found Pareto-optimal designs where both buffers have slow transmission rates (with the fast buffer being slightly faster), while solutions where the fast buffer has a sensibly higher transmission rate, but a proportional packet loss rate, are excluded. In particular, configurations with slow transmission rates have associated good robustness, with very little ranges for objective f_2.

We also observe an interesting relationship between the Pareto-optimal fronts and the Pareto-optimal designs for different values of the sensitivity-awareness parameter $\epsilon \in \{0, 0.05, 0.1\}$. The average values for both objectives f_1 and f_2 experience only little variation as ϵ increases for a fixed tolerance value. For instance, when $\gamma = 0.01$, on average $f_1 \in [369.37, 372.40]$ and $f_2 \in [0.974, 0.98]$, and when $\gamma = 0.05$, $f_1 \in [284.94, 285.48]$ and $f_2 \in [0.995, 0.996]$. Conversely, the average values for the continuous parameters r_slow_rate and delta_rate experience more significant variation and present an interesting negative relationship. More specifically, for any γ value and as the ϵ parameter becomes larger, r_slow_rate shows a decreasing trend while delta_rate shows an increasing trend. We used the Pearson correlation test to analyse this observation and received a strong negative correlation with the coefficient $R \in [-0.992, -0.988]$[4]. This result indicates that as ϵ increases, the sensitivity-aware Pareto-optimal set includes designs in which the transmission rate difference between the slow and fast buffers grows. Although unexpected, this observation is very useful.

Producer-consumer variant. We further analyze a variant of the producer-consumer model, illustrated in Figure 11. In this version, we assume a different redirection strategy (lines 10 and 11) that yields a 100% probability of redirection when the slow buffer is full, while in the original variant the maximum redirection probability is limited to

[4]This result should not be confused with the correlation between the continuous parameters r_slow_rate and delta_rate for fixed γ and ϵ values which ranges from zero to weak, i.e., $R \in [0, 0.3]$.

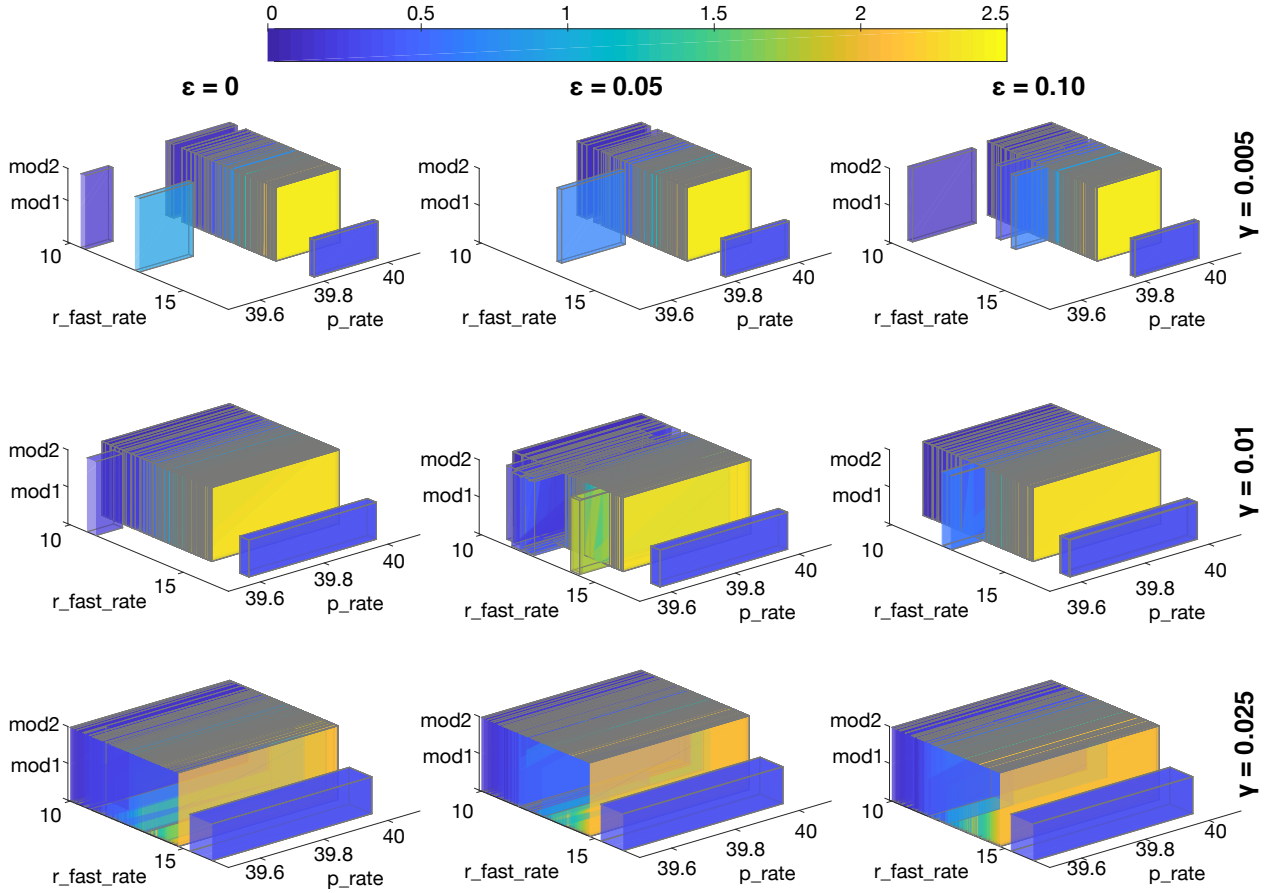


Figure 10: Synthesised Pareto-optimal designs for the second variant of the producer-consumer model and experiments from Figure 9. Rectangles in x-y plane correspond to the continuous parameter regions Boxes are coloured by sensitivity.

0.1. We also consider different continuous parameters: the request production rate (p_rate) and the packet transmission rate for the fast buffer (r_fast_rate). The synthesized Pareto fronts and designs are reported in Figures 9 and 10, respectively.

We observe that the obtained Pareto-optimal set is substantially different from the one obtained in the first variant of the model (Figure 7). Solutions in this variant are generally more robust, demonstrated by the fact that at most one suboptimal solution is synthesised for each configuration. A common trait is that favouring objective f_2 leads to robust designs, while robustness is penalized for high f_1 values. Comparing the two PC variants, whose pCTMC models are shown Figures 3 and 11, we observe that most of the solutions of the second variant are dominated by the Pareto front of the first variant for $\gamma \in \{0.005, 0.01\}$ and all ϵ values, which therefore provides the best performance.

The synthesized parameter regions (Figure 10) confirm the results of the first variant: redirection is always preferred (for all but one design), and the fast buffer rate is not too far from that of the slow buffer (r_fast_rate = 13.03). Similarly, all synthesized values for parameter p_rate are very close to the fixed value (40) used for the same parameter in the first variant of the model. In the

```
1  ctmc
   // buffer capacities
2  const int slow_max = 31;
3  const int fast_max = 21;
4  evolve double p_rate [20..40];      //request production rate
5  const double s_rate = 30;           //request transmission rate
   // packet transmission rates
6  const double r_slow_rate = 10;      //slow buffer
7  evolve double r_fast_rate [10..30]; //fast buffer
8  const double c_rate = 40;           //packet consumption rate
   :
// redirection
9  evolve module Buffer
     :
     //send
10   [] (slow=1) & (buffer_s<slow_max) → s_rate *(1-buffer_s/slow_max) :
                                        (slow'=0) & (buffer_s' = buffer_s +1);
11   [] (slow=1) & (buffer_s>0) & (buffer_f<fast_max) → s_rate *buffer_s/slow_max :
                                        (slow'=0) & (buffer_f' = buffer_f +1);
12   [] (fast=1) & (buffer_f<fast_max) → s_rate :
                                        (fast'=0) & (buffer_f' = buffer_f +1);
     :
13 endmodule
```

Figure 11: Variant of the producer-consumer model introduced in Section 2

Pareto front, we can observe an outlier yielding the highest system throughput (f_1). This design is obtained when

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

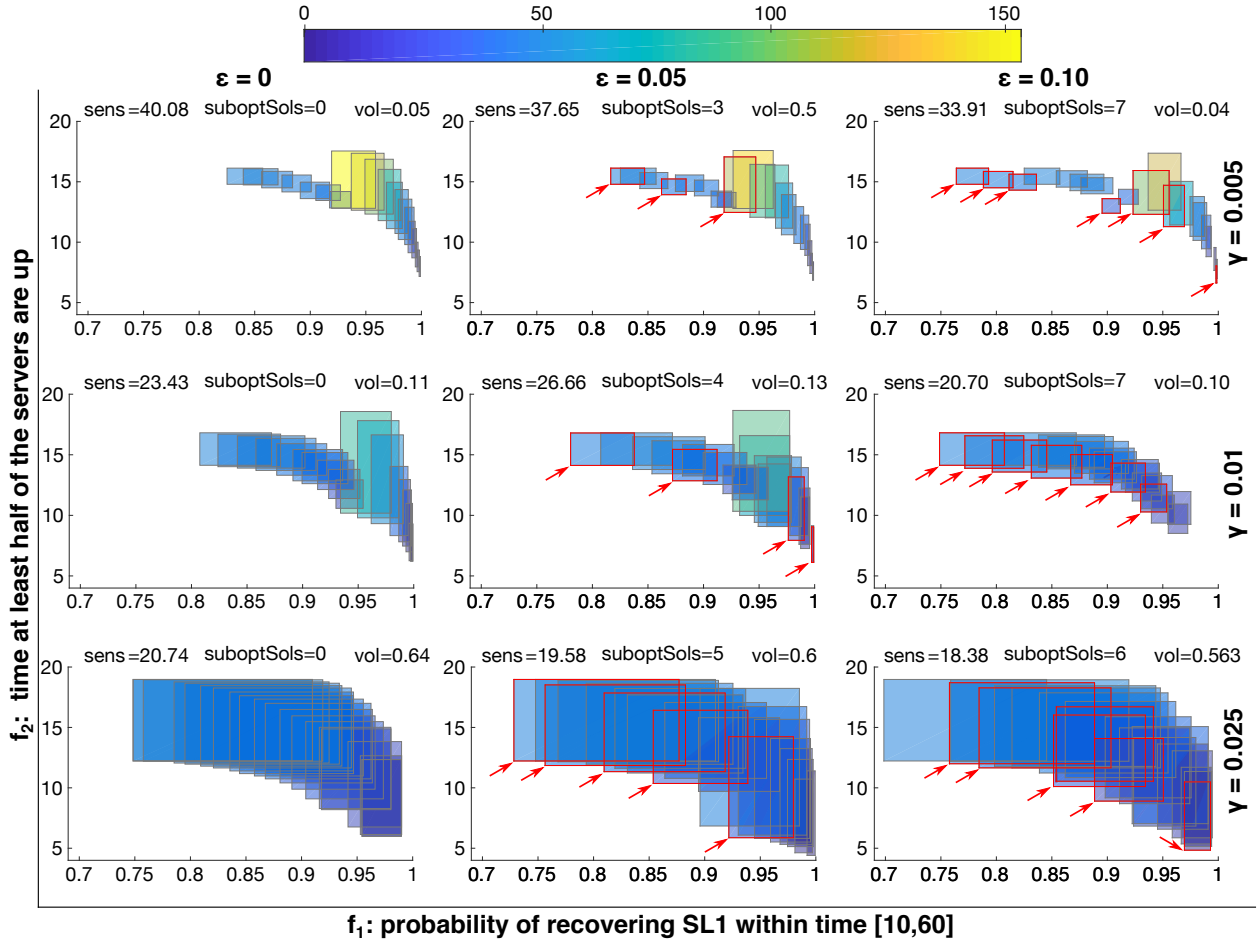


Figure 12: Sensitivity-aware Pareto fronts for the GFS model. Legend and colour code are as in Figure 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

redirection is disabled (see Figure 10). Notably, no other designs with no redirection are present in the Pareto front which provides evidence that redirection is essential to achieve a well-balanced utilisation of the buffers.

Google file system (GFS). Given the pCTMC model, the two maximisation objectives and one constraint of the GFS system, we used RODES to generate Pareto-optimal design sets with tolerances $\gamma \in \{0.005, 0.01, 0.025\}$ for both continuous parameters (cHardFail and cHardRepair) of our pCTMC. Figure 12 shows the Pareto fronts obtained using the "lower bound" definition from Table 1 for the objective functions f_1 and f_2 over candidate designs, and parameters $\epsilon_1 = \epsilon_2 = \epsilon \in \{0, 0.05, 0.1\}$ for the sensitivity-aware Pareto dominance relation (12). The design-space representation is given in Figure 13. We observe that the Pareto front for $\epsilon = 0$ and $\gamma = 0.005$ contains several large (yellow) boxes that correspond to highly sensitive designs. For $\epsilon \in \{0.05, 0.1\}$ and $\gamma = 0.005$, these poor designs are "replaced" by robust designs – surrounded by red borders – with very similar quality attributes but slightly sub-optimal. The same pattern occurs for $\gamma = 0.01$ and (to a lesser extent because of the overall lower sensitivity) for $\gamma = 0.025$. For instance, consider the sensitive design obtained for $\epsilon = 0.1$ and $\gamma = 0.005$ characterized by low hardware fail and repair rates and high number of chunks (yellow bar on Figure 13). Our method found that a more robust solution is possible (highlighted green region), with lower NC and higher cHardFail and cHardRepair.

We also observe that favouring objective f_1 over f_2 generally yields more robust designs (i.e., smaller quality-attribute regions towards the right end of the Pareto fronts) for all combinations of ϵ and γ.

The design-space view of Figure 13 evidences a trade-off between cHardFail and cHardRepair, i.e., optimal designs tend to have either high failure rates and high repair rates, or low failure and repair rates. Results for $\gamma = 0.025$ reveal that there is actually an ideal ratio between the two parameters as the corresponding optimal design appear to keep a relatively constant proportion between cHardFail and cHardRepair. This result was unexpected, yet very useful, since it indicates that designs not satisfying this trade-off yield excessively fast or slow recovery times, and thus are far from the optimal f_1 values.

Further, we observe that the maximum number of chunks per server, NC, has a major influence on the design robustness, with high NC values leading to highly sensitive

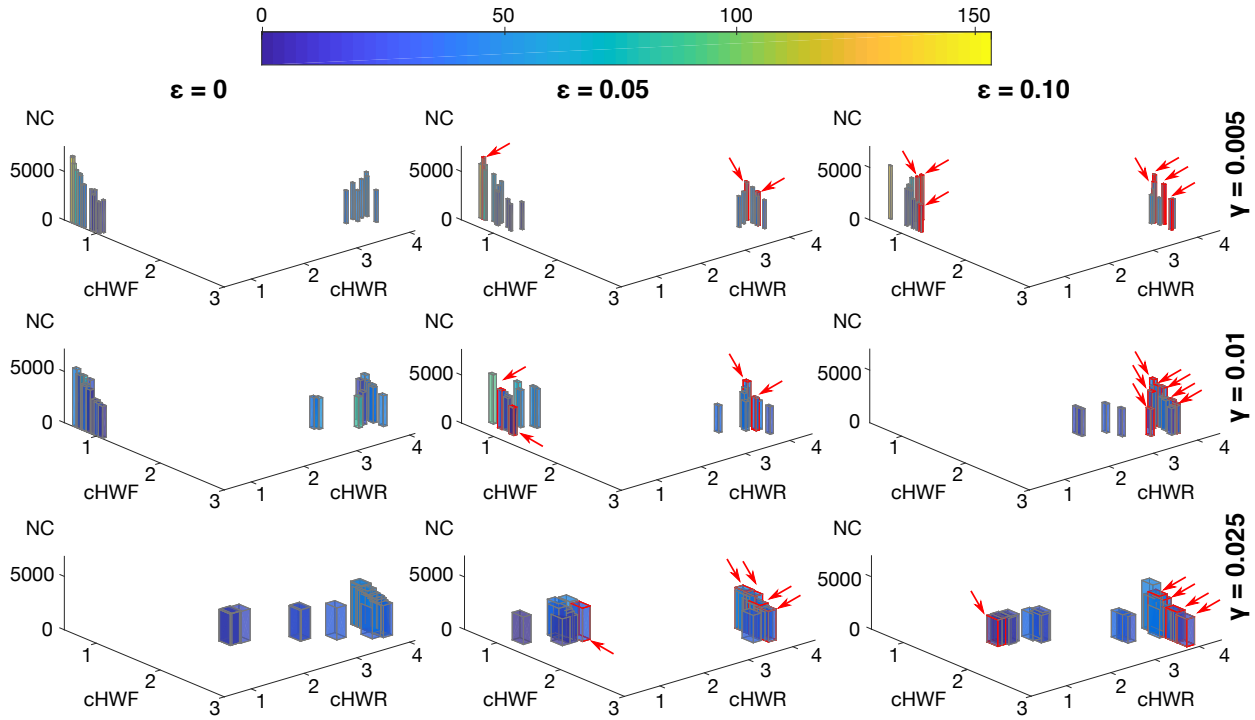


Figure 13: Synthesised Pareto-optimal designs for the GFS model and experiments from Figure 12. Rectangles in x-y plane correspond to the continuous parameter regions.

designs. These designs should be avoided in favour of the alternative designs with low NC values depicted in Figure 13 (for $\epsilon > 0$).

Workstation Cluster (WC). Figure 14 depicts the Pareto fronts obtained for all γ, ϵ combinations of the WC pCTMC model. These Pareto fronts show again how the large quality-attribute regions (corresponding to high-sensitivity designs) obtained for $\epsilon = 0$ are "replaced" by much smaller quality-attribute regions on the Pareto fronts obtained for both $\epsilon > 0$ values. For instance, the fronts produced for $\gamma = 0.005$ and $\epsilon \in \{0.05, 0.10\}$, include sub-optimal robust designs in the objective space $[0.6, 0.8] \times [40, 50]$ that do not exist for $\epsilon = 0$. Further, the Pareto front for $\gamma = 0.005, \epsilon = 0.10$ includes a sub-optimal robust design in the objective space $[0.3, 0.5] \times [45, 70]$ to support the Pareto-optimal but volatile (i.e., highly sensitive) designs within that space. Similar observations can be made for other γ values.

With respect to the system dynamics, our sensitivity-aware synthesis method reveals that the most robust solutions correspond to the objective-function "extrema" from the Pareto front, i.e., to quality-attribute regions in which either f_1 is very high and f_2 is very low, or vice versa. In particular, solutions in the middle of quality-attribute regions are highly sensitive as indicated by the yellow-green boxes for $\gamma = 0.005$ and $\epsilon \in \{0.00, 0.05, 0.10\}$. The equivalent solutions are absent from the Pareto fronts for $\gamma = 0.01$ indicating that they are replaced by more robust solutions whose quality attributes are close to the low- and high-end of their respective ranges. Thus, if designers seek robust solutions they need to select designs that favour one of the quality attributes, since solutions with balanced trade-off between the quality attributes lead to either sensitive or sub-optimal robust designs.

We also identified an interesting property of the synthesized designs. Although they cover the entire design space for the real-valued parameters wsFail, wsCheck and wsRepair, the synthesized designs select very few values for the sub-cluster size N. In particular, in more than 95% of the experiments $N \in \{10, 15\}$ and in the remaining $N \in \{9, 12\}$. We analysed further this observation and ran another experiment by setting the possible range for sub-cluster size $N \in \{11, .., 14\}$. Table 2 compares the average sensitivity between these two experiments for all γ, ϵ combinations. Our results validate that the 'ideal' values of the parameter N for the synthesised robust designs are 10 or 15. This finding demonstrates an unexpected and interesting relationship between the size of the cluster and robustness, impossible to derive through existing analysis methods.

RQ2 (Performance). Since the synthesis process is computationally demanding (see Appendix B), we evaluated the performance of RODES to analyse multiple candidate designs in parallel using the two-level parallelisation architecture described in Section 5.2. By employing the two-level parallelisation, we are able to partially alleviate the CPU overheads incurred not only due to the complexity of evaluating a candidate design but also due to the high number of evaluations. All experiments were run on a CentOS Linux 6.5 64bit server with two 2.6GHz Intel Xeon E5-2670 processors and 64GB memory. For the experiments involving GPU parallelisation, we used two nodes using either an nVidia K40 GPGPU card or an nVidia K80

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

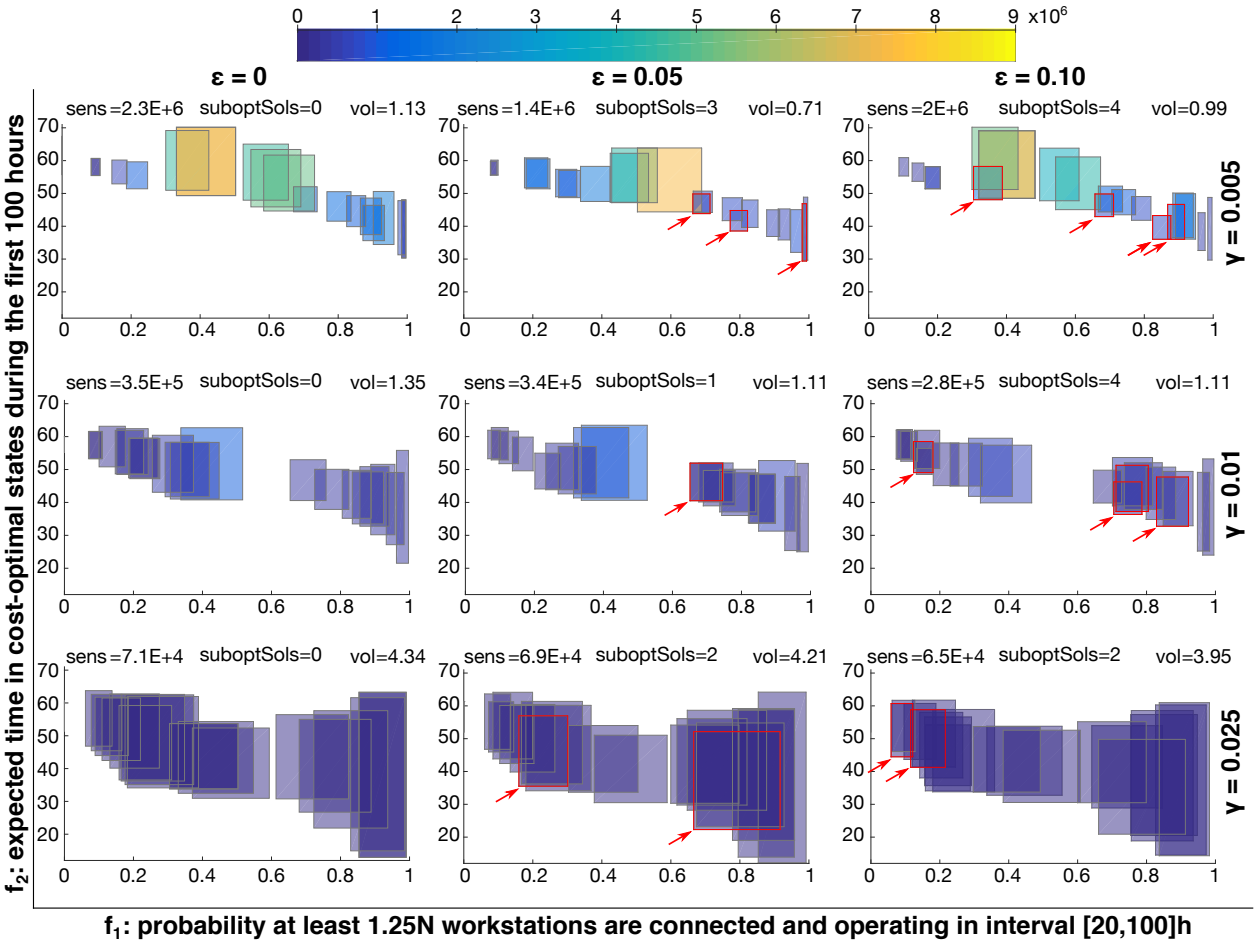


Figure 14: Sensitivity-aware Pareto fronts for the workstation cluster model. Legend and colour code are as in Figure 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

Table 2: Average design sensitivity for two variants of the workstation cluster synthesis problem, given by different ranges for parameter N. Sensitivity-aware designs (i.e. where $\epsilon > 0$) for $N \in \{10..15\}$ have lower sensitivity than for $N \in \{11..14\}$.

	Average sensitivity								
N	$\gamma=0.005$, $\epsilon=0.00$	$\gamma=0.005$, $\epsilon=0.05$	$\gamma=0.005$, $\epsilon=0.10$	$\gamma=0.01$, $\epsilon=0.00$	$\gamma=0.01$, $\epsilon=0.05$	$\gamma=0.01$, $\epsilon=0.10$	$\gamma=0.025$, $\epsilon=0.00$	$\gamma=0.025$, $\epsilon=0.05$	$\gamma=0.025$, $\epsilon=0.10$
{10..15}	1.6E6	7.86E5	6.58E5	2.1E5	2.49E5	2.19E5	6.45E4	6.68E4	7.56E4
{11..14}	1.33E6	1.3E6	1.22E6	5.2E5	5.28E5	4.77E5	2E5	1.93E5	1.87E5

GPGPU card.

The key results of our performance evaluation are described in Tables 3 and 4. The tables show the design synthesis run-times for $k=500$ and $N=20$ (i.e. for $kN=10000$ design evaluations), for our three case studies. Run-time statistics are computed over more than 30 independent runs, obtained using all combinations of $\epsilon \in \{0, 0.05, 0.1\}$ and $\gamma \in \{0.005, 0.01, 0.025\}$. Note that 10000 evaluations, for which we obtained high quality sensitivity-aware Pareto fronts, are still negligible with respect to the size of the design space that an exhaustive search would need to explore (theoretically the design space is uncountable). To demonstrate this difference, we list the number of candidate designs required to "cover" the design space for a given tolerance value γ (this number is indeed much smaller than the total number of candidate designs). For PC model ($\gamma = 0.005$) it is around 20000 designs, but for WC and GFS ($\gamma = 0.01$) it is more than 3 millions designs.

Results in Table 3 confirm that performance of the synthesis process is affected mainly by the size of the underlying pCTMC and by the average number of the discretisation steps required to evaluate particular quantitative attributes (around 4000 steps are required for WC and PC, 160000 for GFS v1, and 46000 for GFS v2). Note that this number depends on the highest time bound appearing in the properties and on the highest rate appearing in the

This is the author's version of an article that has been published in the journal of Systems and Software. Changes were made to this version by the publisher prior to publication. The final version of record is available at https://doi.org/10.1016/j.jss.2018.05.013

Table 3: Time (mean ± SD) in minutes for the synthesis using 10,000 evaluations for one-level CPU parallelisation. **#states** (**#trans.**): number of states (transitions) of the underlying pCTMC. $|K|$: number of continuous parameters.

			CPU (#cores)			
Model	**#states**	**#trans.**	1	2	5	10
WC ($\|K\| = 3$)	3440-8960	18656-49424	394±25	217±29	118±14	68±8
PC ($\|K\| = 2$)	5632	21968-24572	251±46	131±33	50±2	31±5
GFS v1 ($\|K\| = 2$)	1323-2406	7825-15545	390±27	267±49	125±19	71±10
GFS v2 ($\|K\| = 2$)	21606	145335-148245	19011±400	8207±361	4562±36	2399±9

Table 4: Time (mean ± SD) in minutes for the synthesis using 10,000 evaluations for two-level CPU+GPU parallelisation.

	CPU (#cores)			CPU (#cores)/GPU (#devices)				
Model	1	2	5	1/1	2/1	5/1	2/2	5/2
GFS v2	19011±400	8207±361	4562±36	2264±33	1736±8	1625±16	1082±3	1043±22

transition matrix. This observation also explains the significant slowdown of the synthesis process when switching from v1 to v2 of GFS.

First, we evaluate the performance of CPU-only paralellisation at different numbers of cores. The results clearly confirm the scalability with respect to the number of cores. We can also observe that a better scalability is obtained for more complicated synthesis problems (i.e. 5.5-times speed for 10 cores on GFS v1 versus 7.9-times speed up for 10 core on GFS v2).

Second, we evaluate the performance of the two-level parallelisation. Table 4 compares the run-times for different number of CPU cores and GPU devices. In this configuration, we obtain a significant reduction of runtimes, e.g. for GFS v2 we obtain 8.4-times speedup with one GPU and one CPU core, and 7.6-times speedup with two GPUs and two CPU cores. The slightly worse speedup observed in the latter case is due to the increased CPU-GPU communication overhead when more devices are employed.

Finally, we see that evaluating more that one candidate solutions (generated using several CPU cores) on a single GPU further improves the performance until the GPU is fully utilised (i.e. the maximal number of active threads that can be dispatched is reached and thus some parallel evaluations has to be serialised). The performance is also affected by the memory access pattern that depends on the concrete candidate solutions evaluated in parallel. In particular, the performance degrades when the memory access locality is decreased. Note that the maximal number of candidate solutions that can be evaluated in parallel on a single GPU is also limited by the GPU memory that has to accommodate the underlying pCTMC.

RQ3 (Metaheuristic Effectiveness). To answer this research question, we analysed the goodness of the Pareto-optimal designs of the GFS model obtained with our NSGA-II-based RODES against a variant that uses random search (RS). For each variant and combination of $\epsilon \in \{0, 0.05, 0.10\}$ and $\gamma \in \{0.005, 0.01\}$ we carried out 30 independent runs, in line with standard SBSE practice [32]. As building the actual Pareto front for large design spaces is challenging and computationally expensive (GFS has $|\mathcal{P} \times \mathcal{Q}| > 24E10$ assuming a three-decimal precision for continuous parameters), we again followed the standard practice and combined the sensitivity-aware Pareto fronts from all 60 RODES and RS runs for each ϵ, γ combination into a *reference Pareto front* [35]. We then compared the Pareto fronts achieved by each variant against this reference front by using the metrics

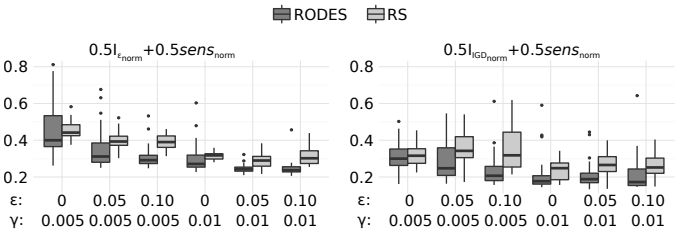


Figure 15: RODES vs. random search (RS) comparison for combinations of $\gamma \in \{0.005, 0.01\}$ and $\epsilon \in \{0, 0.05, 0.10\}$, over 30 independent GFS runs. For both metrics – I_ϵ indicator and sensitivity (left) and I_{IGD} indicator and sensitivity (right) – smaller is better.

$$M_1 = wI_{\epsilon_{norm}} + (1-w)\overline{sens}_{norm}$$

and

$$M_2 = wI_{IGD_{norm}} + (1-w)\overline{sens}_{norm}$$

which use a weight $w \in [0, 1]$ to combine normalised versions of the established (but sensitivity-agnostic) Pareto-front quality metrics I_ϵ and I_{IGD} [35] with the normalised design sensitivity. The unary additive epsilon (I_ϵ) gives the minimum additive term by which the objectives of a particular design from a Pareto front must be altered to dominate the respective objectives from the reference front. The inverted generational distance (I_{IGD}) measures the shortest Euclidean distance from each design in the Pareto front to the closest design in the reference front. These indicators show convergence and diversity to the reference front (smaller is better).

Figure 15 compares RODES and RS across our ϵ, γ combinations using metrics M_1 and M_2 with $w = 0.5$. The

RODES median is consistently lower than that of RS for all ϵ, γ combinations with the exception of $\epsilon = 0, \gamma = 0.01$ (which ignores design sensitivity) for M_2. For a given γ, RODES results improve as ϵ increases, unlike the corresponding RS results. Thus, the difference between RODES and RS increases with larger ϵ for both metrics. This shows that RODES drives the search using sensitivity (10), and thus it can identify more robust designs. We confirmed these visual inspection findings using the non-parametric Mann-Whitney test with 95% confidence level ($\alpha = 0.05$). We obtained statistical significance (p-value < 0.05) for all ϵ, γ combinations except for $\epsilon = 0, \gamma = 0.005$, with p-value in the range [1.71E-06, 0.0026] and [1.086E-10, 0.00061] for M_1 and M_2, respectively.

Considering these results, we have sufficient empirical evidence that RODES synthesises significantly more robust designs than RS. These results are also in line with our previous work which demonstrated through extensive evaluation that probabilistic model synthesis using MOGAs achieves significantly better results that RS [16]. Hence, the problem of synthesising sensitivity-aware Pareto optimal sets (13) is challenging, as expected for any well-defined SBSE problem.

6.5. Threats to Validity.

Construct validity threats may arise due to assumptions made when modelling the three systems. To mitigate these threats, we used models and quality requirements based on established case studies from the literature [36, 34].

Internal validity threats may correspond to bias in establishing cause-effect relationships in our experiments. We limit them by examining instantiations of the sensitivity-aware Pareto dominance relation (12) for multiple values of the sensitivity-awareness ϵ_i and tolerance level γ_k. To alleviate further the risk of biased results due to the MOGAs being stuck at local optimum and not synthesising a global optimum Pareto front, we performed multiple independent runs. Although this scenario never occurred in our experiments, when detected, it can be solved by re-initialising the sub-population outside the Pareto front. Also, Algorithm 1 ensures that the Pareto front monotonically improves at each iteration. Finally, we enable replication by making all experimental results publicly available on the project webpage.

External validity threats might exist if the search for robust designs for other systems cannot be expressed as a pCTMC synthesis problem using objective functions (8) and constraints (9). We limit these threats by specifying pCTMCs in an extended variant of the widely used modelling language of PRISM [21], with objective functions and constraints specified in the established temporal logic CSL. PRISM parametric Markov models are increasingly used to model software architectures, e.g. in the emerging field of self-adaptive software [37, 38, 39, 40]. Another threat might occur if our method generated a Pareto front that approached the actual Pareto front insufficiently, producing only low quality designs or designs that did not satisfy the required quality constraints. We mitigated this threat by using established Pareto-front performance indices to confirm the quality of the Pareto fronts from our case studies. Nevertheless, additional experiments are needed to establish the applicability and feasibility of the method in domains with characteristics different from those used in our evaluation.

7. Related Work

RODES builds on the significant body of *software performance and reliability engineering* research that employs formal models to analyse the quality attributes of alternative software designs, e.g. [5, 6, 7, 8, 9, 10]. Approaches based on formal models such as queueing networks [41], Petri nets [42], stochastic models [43, 44] and timed automata [45, 46], and tools for their simulation (e.g. Palladio [7]) and verification (e.g. PRISM [21] and UPPAAL [45]) have long been used for this analysis. However, unlike RODES, these approaches can only analyse alternative models through a tedious iterative process carried out manually by experts.

Performance antipatterns can be used to speed up this process by avoiding the analysis of poor designs [47, 48, 49], but approaches that automate the search for correct or optimal designs have only been proposed recently. Three types of such approaches are related to RODES. Given a Markov model that violates a quality requirement, the first approach—called *probabilistic model repair* [50, 51]—automatically adjusts its transition probabilities to produce a "repaired" model that meets the requirement. The second approach is called *precise parameter synthesis* [23], and works by identifying transition rates that enable continuous-time Markov models to satisfy a quality requirement or to optimise a quality attribute of the system under development. Finally, our previous work on *probabilistic model synthesis* [16] applies multiobjective optimisation and genetic algorithms to a design template that captures alternative system designs, and generates the Pareto-optimal set of Markov models associated with the quality optimisation criteria of the system. While these approaches represent a significant advance over the previously manual methods of alternative design analysis, they do not take into account the robustness of their repaired or synthesised models. Likewise, the approach from [17] employs evolutionary algorithms to search the configuration space of Palladio Component Models, but the synthesis process does not reflect the sensitivity of the candidate models.

Syntax-guided synthesis has been used to find probabilistic programs that best match the available data [52], including synthesis from "sketches", i.e. partial programs with incomplete details [53]. In [54], counter-example guided inductive synthesis (CEGIS) has been introduced as an

SMT-based synthesiser for sketches and, due to the enormous improvement of SMT solvers in the last decade, CEGIS is currently able to find deterministic programs for a variety of challenging problems [53, 55]. Very recently, the concept of meta-sketches introducing the "optimal synthesis problem" has been proposed [56] and adapted for synthesis of stochastic reaction networks [57]. These solutions are complementary to RODES, as they explore other aspects of design alternatives, and do not take robustness into account.

Methods that rigorously evaluate how the transition probabilities affect the satisfiability of temporal properties (expressed as probabilistic temporal logic formulae) have been studied in the context of parameter synthesis. The methods either construct symbolic expressions describing the satisfaction probability as a function of the model parameters [30, 58], or compute—for given intervals of parameter values—safe bounds on the satisfaction probability [26]. In contrast to this work, our robust design synthesis directly integrates sensitivity analysis into the automated design process.

Another research area related to RODES is *sensitivity analysis*, which analyses the impact of parameter changes on the performance, reliability, cost and other quality attributes of the system under development, e.g. [11, 12, 13]. However, sensitivity analysis typically operates by sampling the parameter space and evaluating the system quality attributes for the sampled values. As such, the result is not guaranteed to reflect the whole range of quality-attribute values for the parameter region of interest. RODES does not have this drawback, as it operates with close over-approximations of the quality-attribute regions for the synthesised robust designs. The perturbation theory for Markov processes has been applied to analysing the sensitivity of software operational profiles [14]. However, this approach quantifies the effect of variations in model transition probabilities without synthesising the analysed solutions. Furthermore, RODES supports a wide range of continuous and discrete parameters that cannot be used with the approach from [14]. Stochastic analysis of architectural models was used for early predictions of system component reliability and sensitivity with respect to different operational profiles [59]. Unlike RODES, the research from [59] focuses on exploiting different architectural models and associated analysis techniques, and is therefore complementary to the work presented in our paper.

The *smoothed model checking* technique from [60] computes an analytical approximation of the satisfaction probability of a formula over a parametric CTMC. While not providing the same guarantees as the safe over-approximation method from RODES, the technique was experimentally shown to be highly accurate, so it can be used to estimate the sensitivity of a probabilistic temporal logic property to variations in the CTMC parameters.

Finally, the problem of parameter synthesis of stochastic reaction networks with respect to multi-objective specification has been recently considered in [61]. The authors employ statistical methods to estimate how kinetic parameters affect the satisfaction probability and average robustness of Signal Temporal Logic properties. In contrast to our approach, a candidate solution from [61] has all parameters fixed and the robustness captures how far the candidate is from violating the particular properties.

8. Conclusion

Robustness is a key and yet insufficiently explored characteristic of software designs, as it can mitigate the unavoidable discrepancies between real systems and their models. We presented RODES, a tool-supported method for the automated synthesis of Pareto-optimal probabilistic models corresponding to robust software designs.

RODES integrates for the first time search-based synthesis and parameter analysis for parametric Markov chains. Our RODES tool automates the application of the method, and provides multi-core as well as GPU-based parallelisation that significantly speeds up the design synthesis process. We performed an extensive experimental evaluation of RODES on three case studies from different application domains. These experiments showed that the sensitivity-aware Pareto-optimal design sets synthesised by RODES enable the selection of robust designs with a wide range of quality-attribute values and provide insights into the system dynamics. The experiments also demonstrate that the parallelisation ensures scalability with respect to the complexity of the systems under development.

In our future work, we will assess the effectiveness of Pareto-dominance relations defined over intervals, and we will augment RODES with alternative multiobjective optimisation techniques such as particle swarm optimisation [62]. In addition, we are planning to extend the RODES modelling language (and the underpinning search method) with support for syntax-based synthesis [63] of robust designs from partial pCTMC specifications, including sketches of chemical reaction networks [57].

References

[1] H. Kitano, Biological robustness, Nature Reviews Genetics 5 (2004) 826–837.

[2] M. S. Phadke, Quality engineering using robust design, Prentice Hall PTR, 1995.

[3] International Organization for Standardization, ISO 286–1:2010, Geometrical product specifications (GPS) – ISO code system for tolerances on linear sizes (2010).
URL https://www.iso.org/standard/45975.html

[4] International Organization for Standardization, ISO general purpose metric screw threads – Tolerances (2013).
URL https://www.iso.org/obp/ui/#iso:std:57778:en

[5] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-based performance prediction in software development: A survey, IEEE Trans. Softw. Eng. 30 (5) (2004) 295–310.

[6] A. B. Bondy, Foundations of Software and System Performance Engineering, Addison Wesley, 2014.

[7] S. Becker, H. Koziolek, R. Reussner, The Palladio component model for model-driven performance prediction, J. Syst. & Softw. 82 (1).

[8] L. Fiondella, A. Puliafito, Principles of Performance and Reliability Modeling and Evaluation, Springer Series in Reliability Engineering, 2016.
[9] W. J. Stewart, Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling, Princeton University Press, 2009.
[10] M. Woodside, D. Petriu, J. Merseguer, D. Petriu, M. Alhaj, Transformation challenges: from software models to performance models, J. Softw. & Syst. Modeling 13 (4) (2014) 1529–1552.
[11] S. S. Gokhale, K. S. Trivedi, Reliability prediction and sensitivity analysis based on software architecture, in: ISSRE'03, 2002, pp. 64–75.
[12] J.-H. Lo, C.-Y. Huang, I.-Y. Chen, et al., Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure, Journal of Syst. and Software 76 (1) (2005) 3 – 13.
[13] C.-Y. Huang, M. R. Lyu, Optimal testing resource allocation, and sensitivity analysis in software development, Transactions on Reliability 54 (4) (2005) 592–603.
[14] S. Kamavaram, K. Goseva-Popstojanova, Sensitivity of software usage to changes in the operational profile, in: NASA Soft. Eng. Workshop, 2003.
[15] A. Filieri, G. Tamburrelli, C. Ghezzi, Supporting self-adaptation via quantitative verification and sensitivity analysis at run time, IEEE Trans. Softw. Eng. 42 (1) (2016) 75–99.
[16] S. Gerasimou, G. Tamburrelli, R. Calinescu, Search-based synthesis of probabilistic models for QoS software engineering, in: ASE'15, 2015, pp. 319–330.
[17] A. Martens, H. Koziolek, S. Becker, R. Reussner, Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, in: WOSP/SIPEW, 2010, pp. 105–116.
[18] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, N. Paoletti, Designing robust software systems through parametric Markov chain synthesis, in: ICSA, IEEE, 2017, pp. 131–140.
[19] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, N. Paoletti, RODES: A robust-design synthesis tool for probabilistic systems, in: QEST, 2017, pp. 304–308.
[20] T. Han, J. Katoen, A. Mereacre, Approximate parameter synthesis for probabilistic time-bounded reachability, in: RTSS, 2008, pp. 173–182.
[21] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of Probabilistic Real-time Systems, in: CAV'11, 2011, pp. 585–591.
[22] M. Kwiatkowska, G. Norman, D. Parker, Stochastic Model Checking, in: SFM'07, 2007, pp. 220–270.
[23] M. Češka, F. Dannenberg, N. Paoletti, et al., Precise parameter synthesis for stochastic biochemical systems, Acta Informatica 54 (2017) 589–623.
[24] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comp. Surveys 45 (1) (2012) 11:1–11:61.
[25] M. Češka, P. Pilař, N. Paoletti, L. Brim, M. Kwiatkowska, PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems, in: TACAS'16, Springer, 2016, pp. 367–384.
[26] T. Quatmann, C. Dehnert, N. Jansen, et al., Parameter synthesis for Markov models: Faster than ever, in: ATVA'16, 2016, pp. 50–67.
[27] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comp. 6 (2) (2002) 182–197.
[28] A. J. Nebro, J. J. Durillo, F. Luna, et al., MOCell: A cellular genetic algorithm for multiobjective optimization, Journal of Intelligent Systems 24 (7) (2009) 726–746.
[29] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[30] C. Dehnert, S. Junges, N. Jansen, et al., PROPhESY: A probabilistic parameter synthesis tool, in: CAV'15, Springer, 2015, pp. 214–231.
[31] C. Dehnert, S. Junges, J.-P. Katoen, M. Volk, A STORM is coming: A modern probabilistic model checker, in: CAV'17, Springer, 2017, pp. 592–600.
[32] M. Harman, P. McMinn, J. de Souza, S. Yoo, Search based software engineering: Techniques, taxonomy, tutorial, in: Emp. Softw. Eng. and Verif., 2012, pp. 1–59.
[33] C. Baier, E. M. Hahn, B. Haverkort, et al., Model checking for performability, Mathematical Structures in Comp. Sc. 23 (4) (2013) 751–795.
[34] B. R. Haverkort, H. Hermanns, J.-P. Katoen, On the use of model checking techniques for dependability evaluation, in: SRDS'00, IEEE, 2000, pp. 228–237.
[35] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, V. da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Trans. Evol. Comp. 7 (2) (2003) 117–132.
[36] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google File System, in: SOSP'03, 2003, pp. 29–43.
[37] R. Calinescu, M. Kwiatkowska, CADS*: Computer-aided development of self-* systems, in: FASE, 2009, pp. 421–424.
[38] R. Calinescu, S. Gerasimou, A. Banks, Self-adaptive software with decentralised control loops, in: FASE, 2015, pp. 235–251.
[39] J. C. Moreno, A. Lopes, D. Garlan, B. Schmerl, Impact models for architecture-based self-adaptive systems, in: FACS, 2015, pp. 89–107.
[40] S. Gerasimou, R. Calinescu, A. Banks, Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration, in: SEAMS, 2014, pp. 115–124.
[41] S. Balsamo, V. D. N. Personè, P. Inverardi, A review on queueing network models with finite capacity queues for software architectures performance prediction, Performance Evaluation 51 (2) (2003) 269–288.
[42] C. Lindemann, Performance modelling with deterministic and stochastic petri nets, Performance Evaluation Review 26 (2) (1998) 3.
[43] R. Calinescu, C. Ghezzi, K. Johnson, et al., Formal verification with confidence intervals to establish quality of service properties of software systems, IEEE Trans. Rel. 65 (1) (2016) 107–125.
[44] V. S. Sharma, K. S. Trivedi, Quantifying software performance, reliability and security: An architecture-based approach, Journal of Systems and Software 80 (4) (2007) 493 – 509.
[45] A. Hessel, K. G. Larsen, M. Mikucionis, et al., Testing real-time systems using UPPAAL, in: Formal methods and testing, Springer, 2008, pp. 77–117.
[46] K. G. Larsen, Verification and performance analysis of embedded and cyber-physical systems using uppaal, in: MODELSWARD'14, 2014, pp. IS–11–IS–11.
[47] D. Arcelli, V. Cortellessa, C. Trubiani, Antipattern-based model refactoring for software performance improvement, in: QoSA'12, 2012, pp. 33–42.
[48] C. U. Smith, L. G. Williams, Software performance antipatterns., in: Workshop on Software and Performance, Vol. 17, 2000, pp. 127–136.
[49] V. Cortellessa, A. Martens, R. Reussner, C. Trubiani, A process to effectively identify 'guilty' performance antipatterns, in: FASE'10, 2010, pp. 368–382.
[50] E. Bartocci, R. Grosu, P. Katsaros, et al., Model repair for probabilistic systems, in: TACAS'11, 2011, pp. 326–340.
[51] T. Chen, E. M. Hahn, T. Han, et al., Model repair for Markov decision processes, in: TASE'13, 2013, pp. 85–92.
[52] A. V. Nori, S. Ozair, S. K. Rajamani, D. Vijaykeerthy, Efficient synthesis of probabilistic programs, in: PLDI'14, 2015, pp. 208–217.
[53] A. Solar-Lezama, R. Rabbah, R. Bodík, K. Ebcioğlu, Programming by sketching for bit-streaming programs, in: Proc. of PLDI'05, ACM, 2005, pp. 281–294.
[54] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, V. Saraswat, Combinatorial sketching for finite programs, in: Proc. of ASPLOS'06, ACM, 2006, pp. 404–415.
[55] A. Solar-Lezama, C. G. Jones, R. Bodik, Sketching concurrent data structures, in: Proc. of PLDI'08, ACM, 2008, pp. 136–148.

[56] J. Bornholt, E. Torlak, D. Grossman, L. Ceze, Optimizing synthesis with metasketches, in: Proc. of POPL'16, ACM, 2016, pp. 775–788.
[57] L. Cardelli, M. Češka, M. Fränzle, M. Kwiatkowska, L. Laurenti, N. Paoletti, M. Whitby, Syntax-guided optimal synthesis for chemical reaction networks, in: CAV'17, Springer, 2017, pp. 375–395.
[58] E. M. Hahn, H. Hermanns, L. Zhang, Probabilistic reachability for parametric Markov models, STTT 13 (1) (2011) 3–19.
[59] L. Cheung, R. Roshandel, N. Medvidovic, L. Golubchik, Early prediction of software component reliability, in: ICSE, ACM, 2008, pp. 111–120.
[60] L. Bortolussi, D. Milios, G. Sanguinetti, Smoothed model checking for uncertain continuous-time markov chains, Inf. Comput. 247 (2016) 235–253.
[61] L. Bortolussi, A. Policriti, S. Silvetti, Logic-based multi-objective design of chemical reaction networks, in: HSB'16, Springer, 2016, pp. 164–178.
[62] M. Reyes-Sierra, C. C. Coello, Multi-objective particle swarm optimizers: A survey of the state-of-the-art, International journal of computational intelligence research 2 (3) (2006) 287–308.
[63] R. Alur, R. Bodik, G. Juniwal, et al., Syntax-guided synthesis, in: FMCAD'13, 2013, pp. 1–8.

Appendix A. Proof of Theorem 1

We show that the sensitivity-aware Pareto dominance relation defined in Defition 4 is a strict order.

Proof. We need to show that relation $\prec$ from (12) is irreflexive and transitive. For any $d \in \mathcal{F}$, $d \prec d$ would require that $f_i(d) < (1+\epsilon_i)f_i(d)$ or $f_i(d) < f_i(d)$ for some $i \in I$, which is impossible. Thus, $\prec$ is irreflexive. To show that $\prec$ is transitive, consider three designs $d, d', d'' \in \mathcal{F}$ such that $d \prec d'$ and $d' \prec d''$. According to (12), we have $\forall i \in I.f_i(d) \leq f_i(d')$ and $\forall i \in I.f_i(d') \leq f_i(d'')$, so $\forall i \in I.f_i(d) \leq f_i(d'')$ due to the transitivity of $\leq$. Furthermore, at least one half of the disjunction from definition (12) must hold for each of $d' \prec d''$ and $d' \prec d''$. We have three cases. Assume first that the left half holds for $d \prec d'$, i.e. that $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d')$ for some $i_1 \in I$; as $f_{i_1}(d') \leq f_{i_1}(d'')$, we also have $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d'')$, so $d \prec d''$ in this case. Assume now that left half of disjunction (12) holds for $d' \prec d''$, i.e., that $(1+\epsilon_{i_1})f_{i_1}(d') < f_{i_1}(d'')$ for some $i_1 \in I$; as $f_{i_1}(d) \leq f_{i_1}(d')$, we again have $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d'')$ and $d \prec d''$. Finally, consider that only the right half of disjunction (12) holds for both $d \prec d'$ and $d \prec d'$. In this last case, $sens(d) \leq sens(d') \leq sens(d'')$ and there is an $i_1 \in I$ such that $f_{i_1}(d) < f_{i_1}(d') \leq f_{i_1}(d'')$, so also $d \prec d''$, and therefore $\prec$ is transitive. □

Appendix B. Complexity Analysis

The time complexity of Algorithm 1 representing the synthesis process is

$$\mathcal{O}\left(k \cdot N \cdot (|I| + |J|) \cdot t + k \cdot |I| \cdot N^2\right),$$

where k is the number of iterations of the (MOGA) while loop (i.e. the number of generations); $N = |CD|$ is the size of the candidate design population; $|I| + |J|$ is the overall number of objective functions and constraints; and t is the time required to analyse a quality attribute of a candidate design. The term $k \cdot N \cdot (|I| + |J|) \cdot t$ quantifies the overall complexity of evaluating candidate designs, while $k \cdot |I| \cdot N^2$ corresponds to comparing designs and building the front in lines 7–14 of Algorithm 1.

The factor t depends on the size of the underlying state space and on the number of discrete-time steps required to evaluate the particular quality attributes. As shown in [23], $t = \mathcal{O}(t_{CSL} \cdot t_{pCSL})$. The factor $t_{CSL} = |\phi| \cdot M \cdot q \cdot t_{\max}$ is the worst-case time complexity of time-bounded CSL model checking [22], where $|\phi|$ is the length of the input CSL formula ϕ, $t_{\max}$ is the highest time bound occurring in it, M is the number of non-zero elements in the rate matrix and q is the highest rate in the matrix. The factor t_{pCSL} is due to the parametric analysis of the design and depends on the form of polynomials appearing in the parametric rate matrix $\mathcal{D}'_{\mathbf{R}}$. Models of software systems typically include only linear polynomials, for which $t_{pCSL} = \mathcal{O}(n)$, where n is the number of continuous parameters.