Designing Robust Software Systems through Parametric Markov Chain Synthesis

Conference Paper · April 2017

DOI: 10.1109/ICSA.2017.16

Radu Calinescu
The University of York

Nicola Paoletti
Stony Brook University

Designing Robust Software Systems through Parametric Markov Chain Synthesis

Radu Calinescu[*], Milan Češka[†], Simos Gerasimou[*], Marta Kwiatkowska[‡] and Nicola Paoletti[§, ‡]
[*]Department of Computer Science, University of York, UK
[†]Faculty of Information Technology, Brno University of Technology, Czech Republic
[‡]Department of Computer Science, University of Oxford, UK
[§]Department of Computer Science, Stony Brook University, USA

***Abstract*—We present a method for the synthesis of software system designs that satisfy strict quality requirements, are Pareto-optimal with respect to a set of quality optimisation criteria, and are robust to variations in the system parameters. To this end, we model the design space of the system under development as a parametric continuous-time Markov chain (pCTMC) with discrete and continuous parameters that correspond to alternative system architectures and to the ranges of possible values for configuration parameters, respectively. Given this pCTMC and required tolerance levels for the configuration parameters, our method produces a sensitivity-aware Pareto-optimal set of designs, which allows the modeller to inspect the ranges of quality attributes induced by these tolerances, thus enabling the effective selection of robust designs. Through application to two systems from different domains, we demonstrate the ability of our method to synthesise robust designs with a wide spectrum of useful trade-offs between quality attributes and sensitivity.**

***Keywords*-software performance and reliability engineering; probabilistic model synthesis; multi-objective optimisation**

I. INTRODUCTION

Evaluating the performance, reliability and other quality attributes of alternative designs is essential for the cost-effective engineering of software [1], [2]. Delaying this evaluation until integration or system testing can greatly increase engineering costs, as defects identified late in the development lifecycle require much more effort to fix [3]. A common method to avoid this delay uses model-based simulation [4] or formal verification [5] to predict the quality attributes of alternative designs. Models that meet the quality requirements of the system under development are then used as a basis for its implementation. Models based on queueing networks [7], probabilistic models [2], [5] and timed automata [8] have been used for this purpose, together with tools for their simulation (e.g. Palladio [9]) and verification (e.g. PRISM [10]). Furthermore, recently proposed approaches automate the search for suitable designs. *Probabilistic model repair* [11], [12] automatically modifies the transition probabilities of Markov models that violate a quality requirement, generating new models that meet the requirement. *Precise parameter synthesis* [13] identifies transition rates that enable continuous Markov models to satisfy a quality requirement or to optimise a quality attribute of the modelled system. Finally, *probabilistic model synthesis* [14] starts from a *design template* that captures alternative system designs, and uses multiobjective optimisation to generate the Pareto-optimal set of Markov models associated with the quality requirements of the system.

However, these promising approaches unrealistically assume that the parameters of the real system (e.g. service rates) will accurately match the parameters of the repaired or synthesised model. This assumption limits the usefulness of existing model repair and synthesis solutions, as Markov models are typically nonlinear, so slight differences between the actual and assumed parameters can lead to major differences between the real and modelled quality attributes of software systems.

Our paper addresses this major limitation for probabilistic model synthesis. To this end, introduce a method for the synthesis of *sensitivity-aware* Pareto-optimal sets of probabilistic models (i.e., *designs*) associated with: (a) the quality requirements of a system; and (b) designer-specified *tolerances* (i.e. permissible levels of variation) in the system parameters. The designs synthesised by our method are *continuous-time Markov chains* with transition rates constrained to bounded intervals that reflect the required tolerances. Accordingly, the Pareto-front element corresponding to each design is a bounded region of quality attribute values for the system. This region is a close over-approximation of all values that the quality attributes can attain for the considered design.

The shape and size of the quality-attribute regions, along with the parameter tolerances, provide key information for sensitivity analysis of the associated Pareto-optimal designs, and thus, for measuring their robustness. In particular, large-tolerance designs associated with small quality-attribute regions are *robust*. Robust designs [15] are of great interest because they can withstand changes in the system parameters, do not expose system users to large variations in quality attributes, and can be built with high-variability components that are often cheaper to develop or purchase, and may require less effort to maintain, than low-variability components. Conversely, large quality-attribute regions from the Pareto front correspond to designs that are highly sensitive to parameter variations, and should typically be avoided.

The main contributions of our paper are threefold. First, we adapt the concept of *tolerance* from other branches of engineering and apply it to software architectures by defining the parametric Markov chain synthesis problem and the sensitivity-aware Pareto dominance relation. Second, we introduce an efficient method that combines probabilistic model synthesis and precise parameter synthesis to automate the generation of sensitivity-aware Pareto fronts for quality engineering. Finally, we present a tool that implements our method for designing robust software systems, which we evaluate on two case studies: a replicated file system used by Google's search engine, and a cluster availability management system. To the best of our knowledge, our work is the first to integrate design synthesis and sensitivity analysis into a single end-to-end method – existing research efforts have tackled the challenges associated with design synthesis (e.g. [14], [16]) and sensitivity analysis (e.g. [17]–[21]) in isolation.

The paper is organised as follows. Sections II and III introduce the background for our work and define the parametric Markov chain synthesis problem, respectively. We then present our robust design synthesis method and tool implementation in Section IV. Finally, we describe the two case studies in Section V, compare our method to related work in Section VI, and conclude the paper with a brief summary in Section VII.

II. Preliminaries

Design space modelling. We use a *parametric continuous-time Markov chain* (pCTMC) to define the design space of the software under development. To this end, we extend the original pCTMC definition [22], where only real-valued parameters determining the transition rates of the Markov chain are considered, and assume that a pCTMC 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 function) 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.

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 $k \in K$;
- $L(q) : S \to 2^{AP}$ 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.

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

$$\mathcal{C}(\mathcal{P}', \{q\}) = (\mathcal{D}'_S, \mathcal{D}'_{init}, \mathcal{D}'_{\mathbf{R}}, L') \tag{2}$$

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 $\gamma_k = \frac{k'^{\top} - k'^{\perp}}{2(k^{\top} - k^{\perp})}$, in line with the fact that the design restricts the value domain for 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.

Quality attribute specification. We specify quality attributes over pCTMCs-defined design spaces using *continuous stochastic logic* (CSL) extended with reward operators [23]. 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:

$$\begin{array}{l} \Phi ::= \mathsf{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\sim r}[\phi] \mid R_{\sim r}[C^{\leq t}] \\ \phi ::= X\ \Phi \mid \Phi\ U^I \Phi \end{array}, \tag{3}$$

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, $t \in \mathbb{R}_{\geq 0}$ 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[2]. As briefly discussed in Section IV-A, our approach can be extended to unbounded CSL.

Traditionally, the semantics of CSL is defined for CTMCs using a satisfaction relation $\models$. Intuitively, a state $s \models P_{\sim r}[\phi]$ 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 $(\omega@t \models \Psi \wedge \forall t' \in [0, t).\omega@t' \models \Phi)$, where $\omega@t$ denotes the state in ω at time t. A state $s \models R_{\sim r}[C^{\leq t}]$ 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 [13], 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 system 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

[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.

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

of a set of CSL path formulas $\{\phi_i\}_{i\in I}$, such that for any $i \in I$ and $(p,q) \in \mathcal{P}\times\mathcal{Q}$,

$$f_i(\mathcal{C}(p,q)) = \Lambda_{\phi_i}(p,q); \tag{4}$$

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 $(p,q) \in \mathcal{P}\times\mathcal{Q}$,

$$c_j(\mathcal{C}(p,q)) \Leftrightarrow \Lambda_{\psi_j}(p,q) \sim_j r_j. \tag{5}$$

Without loss of generality, we will assume that all objective functions $\{f_i\}_{i\in I}$ should be minimised.

III. Synthesis of Parametric Markov Chains

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 $\{c_j\}_{j\in J}$):

$$\begin{aligned}\mathcal{F} = \big\{\mathcal{C}(\mathcal{P}',q) \;\big|\; \mathcal{P}' = \textstyle\mathsf{X}_{k\in K}\left[k'^{\perp}, k'^{\top}\right] \subset \mathcal{P} \wedge q \in \mathcal{Q} \wedge \\ \forall k \in K.\, k'^{\top} - k'^{\perp} = 2\gamma_k(k^{\top} - k^{\perp}) \wedge \\ \forall j \in J. \forall p \in \mathcal{P}'.\, c_j(\mathcal{C}(p,q))\big\}.\end{aligned} \tag{6}$$

The *parametric Markov chain synthesis problem* consists of finding the Pareto-optimal set PS of candidate designs (2) (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 $\{f_i\}_{i\in I}$:

$$\begin{aligned}PS = \big\{\mathcal{C}(\mathcal{P}',q) \in \mathcal{F} \;\big|\; \nexists \mathcal{C}(\mathcal{P}'',q') \in \mathcal{F}. \\ \mathcal{C}(\mathcal{P}'',q') \prec \mathcal{C}(\mathcal{P}',q)\big\},\end{aligned} \tag{7}$$

where the *sensitivity-aware dominance relation* '$\prec$' between two candidate designs is defined below.

Definition 3. 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 $d, d' \in \mathcal{F}$

$$\begin{aligned}d \prec d' \iff \big(\forall i\in I.\, f_i(d) \le f_i(d') \wedge \\ \exists i\in I.\, (1+\epsilon_i) f_i(d) < f_i(d')\big) \vee \big(\forall i\in I.\, f_i(d) \le f_i(d') \\ \wedge\; \exists i\in I.\, f_i(d) < f_i(d') \wedge sens(d) \le sens(d')\big),\end{aligned} \tag{8}$$

where the objective functions $\{f_i\}_{i\in I}$ are now intended over designs $\mathcal{C}(\mathcal{P}',q) \in \mathcal{F}$, and are calculated using one of alternative definitions from Table I; $\epsilon_i \ge 0$ are *sensitivity-awareness parameters*; and 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 $\mathcal{P}'$:

$$sens(\mathcal{C}(\mathcal{P}',q)) = \frac{\prod_{i\in I}\big(f_i^{\top}(\mathcal{C}(\mathcal{P}',q)) - f_i^{\perp}(\mathcal{C}(\mathcal{P}',q))\big)}{\prod_{k\in K} 2\gamma_k(k^{\top} - k^{\perp})}. \tag{9}$$

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

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

TABLE I: 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	$f_i^{\bullet}(\mathcal{C}(\mathcal{P}',q))$	$(f_i^{\perp}(\mathcal{C}(\mathcal{P}',q)) + f_i^{\top}(\mathcal{C}(\mathcal{P}',q)))/2$

Proof. We need to show that relation $\prec$ from (8) 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 (8), we have $\forall i \in I. f_i(d) \le f_i(d')$ and $\forall i \in I. f_i(d') \le f_i(d'')$, so $\forall i \in I. f_i(d) \le f_i(d'')$ due to the transitivity of $\le$. Furthermore, at least one half of the disjunction from definition (8) 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 $f_{i_1}(d') \le f_{i_1}(d'')$, 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 (8) 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 $f_{i_1}(d) \le f_{i_1}(d')$, 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 (8) holds for both $d \prec d'$ and $d \prec d'$. In this last case, $sens(d) \le sens(d') \le sens(d'')$ and there is an $i_1 \in I$ such that $f_{i_1}(d) < f_{i_1}(d') \le f_{i_1}(d'')$, so also $d \prec d''$, and therefore $\prec$ is transitive. $\square$

The classical Pareto dominance definition can be obtained by setting $\epsilon_i = 0$ for all $i \in I$ in definition (8). 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. $sens(d) \le sens(d')$.

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 I captures specific needs of real-world systems. Thus, using the "upper bound" definition ($f_i^{\top}$) in (8) 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 I ($f_i^{\perp}$) 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 (9)—to compare and select designs based on their reference midpoint quality values.

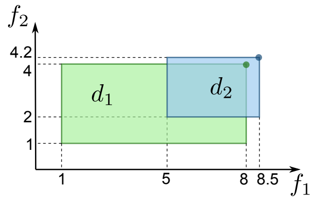

Fig. 1: Quality-attribute regions for designs d_1, d_2.

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. This is illustrated in Fig. 1 for the quality-attribute regions induced by two potential pCTMC designs d_1, d_2 (which we assume associated with identical parameter tolerances and thus, same parameter space volume V) and two minimisation objectives f_1, f_2. In this scenario, using $f_i = f_i^\top$ in (8), we have $d_1 \prec d_2$ when $\epsilon_1 = \epsilon_2 = 0$ (classical dominance) because $f_1^\top(d_1) = 8 < 8.5 = f_1^\top(d_2)$ and $f_2^\top(d_1) = 4 < 4.2 = f_2^\top(d_2)$, but $d_1 \not\prec d_2$ when $\epsilon_1 = \epsilon_2 = 0.1$ (so d_2 is retained in the sensitivity-aware Pareto-optimal set) because $1.1 \cdot f_1^\top(d_1) \not< f_1^\top(d_2)$, $1.1 \cdot f_2^\top(d_1) \not< f_2^\top(d_2)$ and $sens(d_1) \not\leq sens(d_2)$ because $sens(d_1) = ((8-1)\cdot(4-1))/V$ and $sens(d_2) = ((8.5-5)\cdot(4.2-2))/V$.

IV. Parametric CTMC Synthesis Method

Computing the Pareto-optimal design set (7) is typically unfeasible, as the design space $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ is infinite 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 [13], [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 3–12 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 5–12. 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 constraints (line 7), the for loop in lines 9-11 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 12).

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

Algorithm 1 Parametric Markov chain synthesis

1: **function** $\textsc{Synthesis}(\mathcal{C}(\mathcal{P},\mathcal{Q}), \{f_i\}_{i\in I}, \{c_j\}_{j\in J}, \{\gamma_k\}_{k\in K})$
2: $\quad \overline{PS} \leftarrow \emptyset$
3: $\quad$ **while** $\neg\textsc{Terminate}(\mathcal{C}(\mathcal{P}, \mathcal{Q}), \overline{PS})$ **do**
4: $\quad\quad CD \leftarrow \textsc{CandidateDesigns}(\mathcal{C}(\mathcal{P}, \mathcal{Q}), \{\gamma_k\}_{k\in K}, \overline{PS})$
5: $\quad\quad$ **for all** $d \in CD$ **do**
6: $\quad\quad\quad (\{f_{i,d}^\top\}_{i\in I}, \{f_{i,d}^\perp\}_{i\in I}, \{c_{j,d}\}_{j\in J}) \leftarrow \textsc{AnalyseDesign}(d, \{f_i\}_{i\in I}, \{c_j\}_{j\in J})$
7: $\quad\quad\quad$ **if** $\bigwedge_{j\in J} c_{j,d}$ **then**
8: $\quad\quad\quad\quad dominated = \mathsf{false}$
9: $\quad\quad\quad\quad$ **for all** $d' \in \overline{PS}$ **do**
10: $\quad\quad\quad\quad\quad$ **if** $d' \prec d$ **then** $dominated = \mathsf{true}$; **break**
11: $\quad\quad\quad\quad\quad$ **if** $d \prec d'$ **then** $\overline{PS} = \overline{PS} \setminus \{d'\}$
12: $\quad\quad\quad\quad$ **if** $\neg dominated$ **then** $\overline{PS} = \overline{PS} \cup \{d\}$
13: $\quad$ **return** $\overline{PS}$

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

A. Computing Safe Property Bounds for pCTMCs

To establish the quality attributes and sensitivity of candidate designs, $\textsc{AnalyseDesign}$ uses precise parameter synthesis techniques [13] 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 $\Lambda_{\min}^q$ and a safe over-approximation $\Lambda_{\max}^q$ of the minimal and maximal probability that $\mathcal{C}(\mathcal{P}', q)$ satisfies ϕ:

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

This allows us to safely approximate the bounds $\{f_i^\perp, f_i^\top\}_{i\in I}$ of the objective functions, and the constraints $\{c_j\}_{j\in J}$. As shown in [13], the over-approximation quality improves as the size of $\mathcal{P}'$ decreases, and thus can be effectively controlled.

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 [23] 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 minimisation/maximisation of state probabilities in a time non-homogenous Markov process. Presenting the technique in detail is outside the scope of our paper, but the interested reader can find a complete description in [13].

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.

B. Metaheuristic for Parametric CTMC Synthesis

To ensure that CANDIDATEDESIGNS selects suitable candidate designs, Algorithm 1 is implemented as an established *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 $\mathcal{P}'$.

Given this encoding of candidate designs, the first execution of CANDIDATEDESIGNS from Algorithm 1 returns a randomly generated *population* (i.e. set) of feasible designs (6). 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 (8) 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. 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. Due to space constraints, we do not provide these details, which are available in [27] for the NSGA-II MOGA used in our experimental evaluation from Section V.

Complexity. The time complexity of 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 in Algorithm 1; $N = |CD|$ 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 7–12 of Algorithm 1. Increasing the total number of design evaluations (i.e., $k \cdot N$) typically improves the Pareto optimality of the resulting design set, but also slows down the synthesis process.

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 [13], $t = \mathcal{O}(t_{CSL} \cdot t_{pCSL})$. The factor $t_{CSL} = |\phi| \cdot M \cdot q \cdot t_{\max}$ is the worst-case time complexity of time-bounded CSL model checking [23], 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.

C. Implementation

We developed a Java software tool that implements the pCTMC synthesis method from Algorithm 1. For its ANALYSEDESIGN function, we used PRISM-PSY [25], a verification engine that supports precise parameter synthesis by efficient parametric backward transient analysis. We realised the functionality of CANDIDATEDESIGNS using the jMetal [30] Java framework for multi-objective optimisation with metaheuristics. Our Robust DEsign Synthesis (RODES) tool operates with pCTMCs expressed in the high-level modelling language of PRISM [10] extended with the following constructs (adopted from [14]) for specifying the parameters $k \in K$ and $d \in D$ from Definition 1:

$$\begin{array}{l} \textsf{evolve double } k \ [min..max] \\ \textsf{evolve int } d \ [min..max] \\ \textsf{evolve module } ComponentName \end{array} \tag{10}$$

$N > 1$ 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. The open-source code of RODES, supplementary material on the case studies and the full experimental results are available on our project website at http://www-users.cs.york.ac.uk/~simos/RODES.

V. CASE STUDIES

We evaluated our design synthesis method in two case studies from different domains, using the RODES tool with the following NSGA-II configuration: 10000 evaluations, initial population 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.

Google File System (GFS). Our first case study considers the design of GFS, the replicated file system used by Google's search engine [31]. 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) [32]. In our work, we adapt the CTMC of the lifecycle of a GFS chunk from [32] by considering several continuous and discrete parameters that a designer of the

```
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
```

Fig. 2: pCTMC model of the Google File System

system has to decide. Fig. 2 shows the resulting model, encoded in the PRISM modelling language extended with the evolve constructs from Section IV-C. As in [32], 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 our method, 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 Fig. 2. 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: $\phi_1 = \neg \mathsf{SL1}\ U^{[10,60]}\ \mathsf{SL1}$, 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: $\phi_2 = C^{\leq 60}$, 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: $\psi_1 = C^{\leq 60}$ with threshold $\sim_1 r_1 \equiv$ '≤ 5', 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.

Given these objective functions and constraint, and the GFS pCTMC, we used our RODES tool from Section IV-C to generate Pareto-optimal design sets for the GFS system, with tolerances $\gamma \in \{0.01, 0.02, 0.05\}$ for both continuous parameters (cHardFail and cHardRepair) of our pCTMC. Fig. 3 shows the Pareto fronts obtained using the "lower bound" definition from Table I 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 (8). These Pareto fronts provide a wealth of information supporting the evaluation of the optimality and robustness of alternative GFS designs. In particular, the Pareto front for $\epsilon = 0$ and $\gamma = 0.01$ contains several large (red) boxes that correspond to highly sensitive designs. For $\epsilon \in \{0.05, 0.1\}$ and $\gamma = 0.01$, these poor designs are "replaced" by robust designs – surrounded by (red) borders – with very similar quality attributes but slightly suboptimal. The same pattern occurs for $\gamma = 0.02$ and (to a lesser extent because the sensitivity (9) decreases when the tolerance grows) for $\gamma = 0.05$. 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.

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. For fixed ϵ, increasing the parameter tolerance γ leads, as expected, to larger (more uncertain) quality-attribute regions and, typically, to an improved robustness (as explained above).

The corresponding synthesised sensitivity-aware Pareto-optimal designs provide key insights into the GFS dynamics, as shown in Fig. 4 for several ϵ, γ combinations and fully on our project website. While for $\epsilon = 0$ we obtain only optimal solutions when parameters cHardFail and cHardRepair assume their extreme values, adding sensitivity leads to additional designs that are close to the optimum and at the same time are significantly more robust. These designs appear along an "ideal diagonal" in the horizontal plane suggesting the presence of an optimal ratio between cHardFail and cHardRepair: designs outside this diagonal yield excessively fast or slow recovery times, and thus are far from the optimal f_1 values. Further,

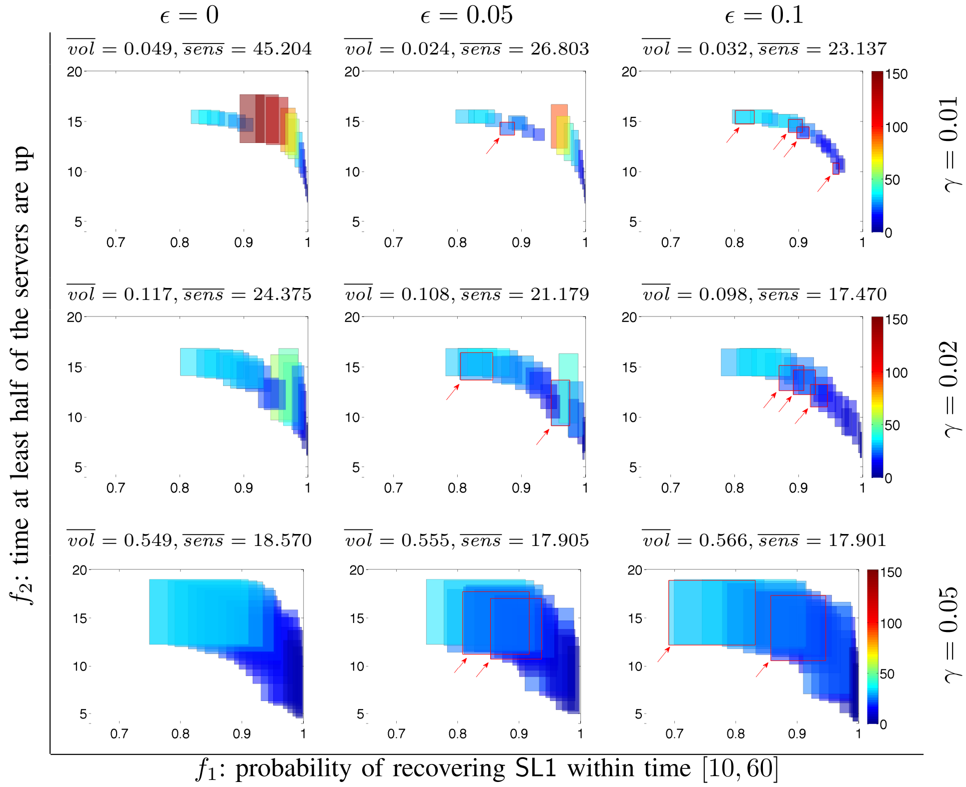

Fig. 3: Sensitivity-aware Pareto fronts for the GFS model over objectives f_1 (x-axis) and f_2 (y-axis). Boxes represent quality-attribute regions, coloured by sensitivity (red: sensitive, blue: robust). Red-bordered boxes and arrows indicate the sub-optimal robust designs. For each front, we report mean sensitivity ($\overline{sens}$) and mean volume ($\overline{vol}$).

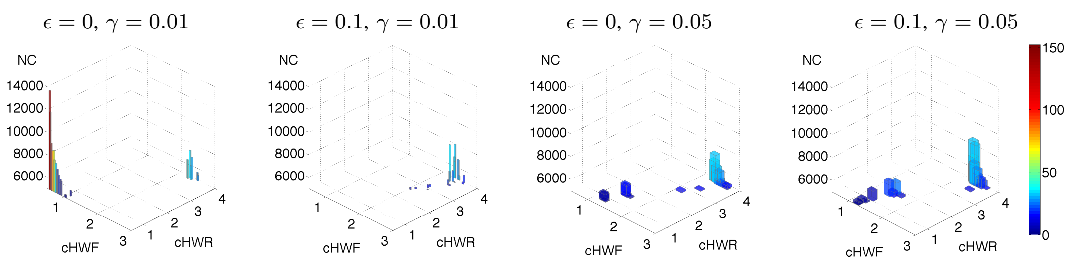

Fig. 4: Synthesised Pareto-optimal designs for the GFS model and experiments from Fig. 3. Rectangles in x-y plane correspond to the continuous parameter regions (cHWF: hardware failure rate; cHWR: hardware repair rate) induced by the tolerance γ. The box heights show the value of the discrete parameter NC. Boxes are coloured by sensitivity.

our method reveals 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 designs. These designs should be avoided in favour of the alternative designs with low NC values depicted in Fig. 4 (for $\epsilon > 0$).

We analysed the goodness of the Pareto-optimal designs obtained with our NSGA-II-based RODES against a tool variant that uses random search (RS). For each tool variant and combination of $\epsilon \in \{0, 0.05, 0.10\}$ and $\gamma \in \{0.01, 0.02\}$ we carried out 30 independent runs, in line with standard SBSE practice [33]. As building the actual Pareto front for large design spaces is unfeasible, 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*. We then compared the Pareto fronts achieved by each variant against this reference front by using the metrics $M_1 = wI_{\epsilon_{norm}} + (1-w)\overline{sens}_{norm}$ 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} [33] with the normalised design sensitivity.[3] Fig. 5 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

[3] 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. The indicators measure convergence to the reference front and design diversity.

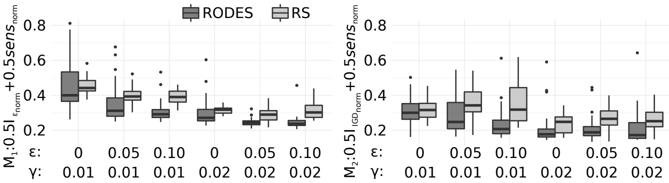

Fig. 5: RODES vs. random search (RS) comparison for combinations of $\gamma \in \{0.01, 0.02\}$ and $\epsilon \in \{0, 0.05, 0.10\}$, over 30 independent GFS runs. For both metrics – I_ϵ indicator and sensitivity (left) and I_{IGD} indicator and sensitivity (right) – smaller is better.

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 (9), 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 $\epsilon = 0, \gamma = 0.01$, with p-value in the range [1.71E-06, 0.0026] and [1.086E-10, 0.00061] for M_1 and M_2, respectively.

Workstation Cluster (WC) For this case study 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, wsFail and $\mathsf{wsRepair}$ 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. This model is illustrated on our project website. 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 and one constraint:

- f_1: $\phi_1 = \neg\mathsf{premium}\ U[20, 100]\ \mathsf{premium}$ where $\mathsf{premium}$ denotes a system service where at least $1.25N$ workstations are connected and operating;
- f_2: $\phi_2 = C^{\leq 100}$ where a reward of 1 is assigned to states with a number of operating clusters between $1.2N$ and $1.6N$;
- c_1: $\psi_1 = C^{\leq 100}$ with threshold $\sim_1 r_1 \equiv$ '≤ 80', where transition rewards are associated with repair actions of the workstations, switches and backbone.

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

Due to space constraints, we include only the Pareto fronts obtained for a tolerance level $\gamma = 0.01$ for all real-valued system parameters, and for a sensitivity-awareness parameter $\epsilon \in \{0, 0.05, 0.1\}$ for both objective functions (Fig. 6, top). These Pareto fronts show again how increasing ϵ yields significant gains in design robustness, with mean sensitivity values for $\epsilon = 0.05$ and $\epsilon = 0.1$ that are 51% and 59% smaller than the mean sensitivity for $\epsilon = 0$, respectively. Visual inspection confirms that 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.

With respect to the system dynamics, our sensitivity-aware synthesis 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. We further observe and validate (Fig. 6, bottom and Table II) that the values of the parameter N for the synthesised robust designs are 10 or 15. This shows an unexpected and interesting relationship between the size of the cluster and robustness, impossible to derive through existing analysis methods.

Performance. As the design synthesis is computationally demanding, the current RODES version analyses multiple candidate models in parallel using multi-core architectures. In this way, we are able to partially alleviate the burden related to the high number of evaluations. Table III shows the design synthesis run-times for $k = 500$ and $N = 20$ (i.e. for $kN = 10000$ design evaluations), for several variants of our case studies corresponding to different discrete parameter values (and thus to different pCTMC sizes). Run-time statistics are computed over 9 independent experiments each, given by all combinations of $\gamma \in \{0.01, 0.02, 0.05\}$ and $\epsilon \in \{0, 0.05, 0.1\}$. The synthesis time varies between 6262.22s for the smallest system instance (GFS, S=5000) and 12295.55s for the largest instance (WC, N=15). The average synthesis time over all scenarios is 7123.6s for the GFS case study and 11208.8s for WC, confirming that performance is affected by the size of the underlying pCTMC and the number of continuous parameters.

All the experiments of this section were run on a CentOS Linux 6.5 64bit server with two 2.6GHz Intel Xeon E5-2670 processors and 32GB memory, and reported run-times were obtained using multi-core parallelisation. In the oncoming version of the tool we plan to integrate the GPU-accelerated precise parameter synthesis methods of [25], which would significantly improve the scalability of the synthesis process with respect to the size of the candidate designs.

Threats to Validity. *Construct validity* threats may arise due to assumptions made when modelling the two systems. To mitigate these threats, we used models and quality requirements based on established case studies from the literature [31], [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-

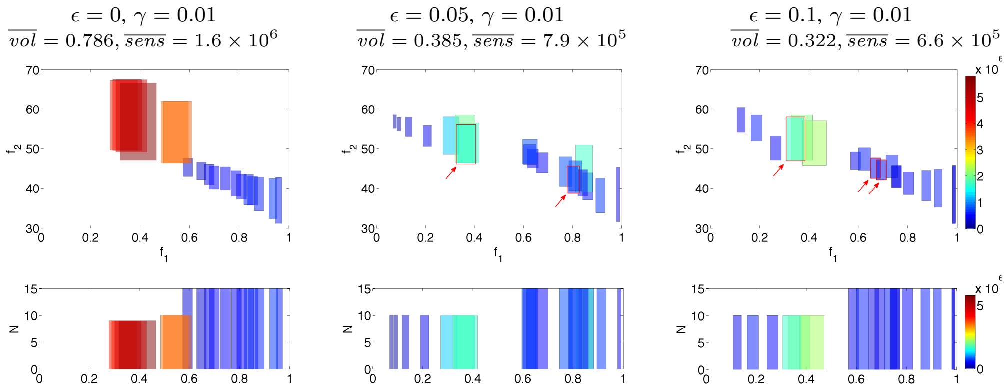

Fig. 6: *Top:* sensitivity-aware Pareto fronts for the cluster model over objectives f_1 (x-axis) and f_2 (y-axis). *Bottom:* "projection" of the quality-attribute regions on the f_1-N plane corresponding to synthesised sensitivity-aware designs. Colour code, $\overline{sens}$ and $\overline{vol}$ are as in Fig. 3.

TABLE II: 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	γ=0.01, ϵ=0.00	γ=0.01, ϵ=0.05	γ=0.01, ϵ=0.10	γ=0.02, ϵ=0.00	γ=0.02, ϵ=0.05	γ=0.02, ϵ=0.10	γ=0.05, ϵ=0.00	γ=0.05, ϵ=0.05	γ=0.05, ϵ=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

TABLE III: Time (mean $\pm$ SD) for the synthesis using 10,000 evaluations. **Scenario**: values of discrete parameters. **#states** (**#trans.**): number of states (transitions) of the underlying pCTMC. $|K|$: number of continuous parameters.

System	**Scenario**	**#states**	**#trans.**	**Time (s)**
Google File System ($\|K\|$=2)	S=5000	1323	7825	6262.22 $\pm$ 236.26
	S=10000	1893	11843	8943.33 $\pm$ 243.05
	S=20000	2406	15545	10818.89 $\pm$ 539.73
Workstation Cluster ($\|K\|$=3)	N=9	3440	18656	11080.5 $\pm$ 1165.17
	N=12	5876	32204	11451.11 $\pm$ 1597.93
	N=15	8960	49424	12295.55 $\pm$ 2535.12

aware Pareto dominance relation (8) 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. 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 (4) and constraints (5). We limit these threats by specifying pCTMCs in an extended variant of the widely used modelling language of PRISM [10], 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 [**?**], [**?**], [**?**], [**?**]. 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.

VI. Related Work

In previous work [14], we proposed a purely search-based engineering approach that uses evolutionary algorithms to synthesise probabilistic models that satisfy multi-objective specifications. However, the designs generated by this approach are non-parameteric probabilistic models, and thus cannot support sensitivity analysis like our new method. Similarly, the research from [16] employs evolutionary algorithms to search the configuration space of Palladio Component Models, but does not consider the sensitivity of the obtained models.

Sensitivity analysis has long been used to assess the impact that changes in the parameters of the system under development have on the system performance, reliability and other quality attributes, e.g. in [17]–[19]. However, these approaches work by repeatedly sampling the parameter space of the system and evaluating the system behaviour for the sampled values. Accordingly, their results are not guaranteed to capture the entire range of quality-attribute values for the parameter region of interest. Our method overcomes this limitation by generating safe and close over-approximations of the quality-attribute regions associated with robust designs.

The sensitivity of software operational profiles has been analysed using the perturbation theory for Markov processes [20], to quantify the effect of variations in model transition probabilities. However, this approach does not synthesise the solutions, and does not work with the wide range of continuous and discrete parameters supported by our method.

Finally, research on parameter synthesis for probabilistic systems from temporal logic specifications focuses on deriving symbolic expressions for the satisfaction probability of the specification as a function of the parameters [21], [35], [36] or on computing safe enclosures of the satisfaction probability for given intervals of parameter values [13], [26]. In contrast to this work, our robust design synthesis directly integrates sensitivity analysis into the automated design process.

VII. Conclusion

The analysis of model sensitivity is key for effective design automation, as it establishes how models are affected by parameter deviations, accounting for the unavoidable discrepancies between the real systems and their models. We presented a method for the automated synthesis of Pareto-optimal and robust software designs, which builds on search-based synthesis and parameter synthesis for parametric Markov chains. We developed a tool that implements the method and we used it in two case studies, showing that our synthesised sensitivity-aware Pareto-optimal design sets support the selection of robust designs with a wide range of quality-attribute values and provide insights into the system dynamics.

As future work, we plan to investigate Pareto-dominance relations defined over intervals; alternative search techniques (e.g. particle swarm optimisation [38]); and extensions of the modelling language and the search method to support syntax-based synthesis [39] of robust designs from partial/incomplete pCTMC specifications.

Acknowledgments

This work has been partially supported by the Czech Grant Agency grant No. GA16-17538S (M. Češka).

References

[1] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *ICSE*, 2008, pp. 111–120.

[2] V. S. Sharma and K. S. Trivedi, "Quantifying software performance, reliability and security: An architecture-based approach," *Journal of Systems and Software*, vol. 80, no. 4, pp. 493 – 509, 2007.

[3] L. O. Damm and L. Lundberg, "Company-wide implementation of metrics for early software fault detection," in *ICSE*, 2007, pp. 560–570.

[4] F. Brosig, P. Meier, S. Becker *et al.*, "Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures," *IEEE Trans. Softw. Eng.*, vol. 41, no. 2, pp. 157–175, 2015.

[5] 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.*, vol. 65, no. 1, pp. 107–125, 2016.

[6] J. M. Franco, R. Barbosa, and M. Zenha-Rela, "Automated reliability prediction from formal architectural descriptions," in *ECSA'12*, 2012.

[7] S. Balsamo, V. D. N. Personè, and P. Inverardi, "A review on queueing network models with finite capacity queues for software architectures performance prediction," *Performance Evaluation*, vol. 51, no. 2, pp. 269–288, 2003.

[8] A. Hessel, K. G. Larsen, M. Mikucionis *et al.*, "Testing real-time systems using uppaal," in *Formal methods and testing*, 2008, pp. 77–117.

[9] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *J. Syst. & Softw.*, vol. 82, no. 1, 2009.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," in *CAV'11*, 2011, pp. 585–591.

[11] E. Bartocci, R. Grosu, P. Katsaros *et al.*, "Model repair for probabilistic systems," in *TACAS'11*, 2011, pp. 326–340.

[12] T. Chen, E. M. Hahn, T. Han *et al.*, "Model repair for Markov decision processes," in *TASE'13*, 2013, pp. 85–92.

[13] M. Češka, F. Dannenberg, N. Paoletti *et al.*, "Precise parameter synthesis for stochastic biochemical systems," *Acta Informatica*, pp. 1–35, 2016.

[14] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for QoS software engineering," in *ASE'15*, 2015, pp. 319–330.

[15] M. Phadke, *Quality Engineering Using Robust Design*. Prentice Hall, 1995.

[16] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *WOSP/SIPEW*, 2010, pp. 105–116.

[17] S. S. Gokhale and K. S. Trivedi, "Reliability prediction and sensitivity analysis based on software architecture," in *ISSRE'03*, 2002, pp. 64–75.

[18] 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*, vol. 76, no. 1, pp. 3 – 13, 2005.

[19] C.-Y. Huang and M. R. Lyu, "Optimal testing resource allocation, and sensitivity analysis in software development," *Transactions on Reliability*, vol. 54, no. 4, pp. 592–603, 2005.

[20] S. Kamavaram and K. Goseva-Popstojanova, "Sensitivity of software usage to changes in the operational profile," in *NASA Soft. Eng. Workshop*, 2003.

[21] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Trans. Softw. Eng.*, vol. 42, no. 1, pp. 75–99, 2016.

[22] T. Han, J. Katoen, and A. Mereacre, "Approximate parameter synthesis for probabilistic time-bounded reachability," in *RTSS*, 2008, pp. 173–182.

[23] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic Model Checking," in *SFM'07*, 2007, pp. 220–270.

[24] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comp. Surveys*, vol. 45, no. 1, pp. 11:1–11:61, 2012.

[25] M. Češka, P. Pilař, N. Paoletti, L. Brim, and M. Kwiatkowska, "PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems," in *TACAS'16*, 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, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comp.*, vol. 6, no. 2, pp. 182–197, 2002.

[28] A. J. Nebro, J. J. Durillo, F. Luna *et al.*, "MOCell: A cellular genetic algorithm for multiobjective optimization," *Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.

[29] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[30] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011.

[31] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *SOSP'03*, 2003, pp. 29–43.

[32] C. Baier, E. M. Hahn, B. Haverkort *et al.*, "Model checking for performability," *Mathematical Structures in Comp. Sc.*, vol. 23, no. 4, pp. 751–795, 2013.

[33] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Trans. Evol. Comp.*, vol. 7, no. 2, pp. 117–132, 2003.

[34] B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "On the use of model checking techniques for dependability evaluation," in *SRDS'00*, 2000.

[35] C. Dehnert, S. Junges, N. Jansen *et al.*, "PROPhESY: A probabilistic parameter synthesis tool," in *CAV'15*, 2015, pp. 214–231.

[36] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," *STTT*, vol. 13, no. 1, pp. 3–19, 2011.

[37] H. Karshenas, C. Bielza, and P. Larrañaga, "Interval-based ranking in noisy evolutionary multi-objective optimization," *Computational Optimization and Applications*, vol. 61, no. 2, pp. 517–555, 2015.

[38] M. Reyes-Sierra and C. C. Coello, “Multi-objective particle swarm optimizers: A survey of the state-of-the-art,” *International journal of computational intelligence research*, vol. 2, no. 3, pp. 287–308, 2006.
[39] R. Alur, R. Bodik, G. Juniwal *et al.*, “Syntax-guided synthesis,” in *FMCAD’13*, 2013, pp. 1–8.