Reducing Run-Time Adaptation Space via Analysis of Possible Utility Bounds

Clay Stevens
University of Nebraska-Lincoln
Department of Computer Science and Engineering
clay.stevens@huskers.unl.edu

Hamid Bagheri
University of Nebraska-Lincoln
Department of Computer Science and Engineering
bagheri@unl.edu

ABSTRACT

Self-adaptive systems often employ dynamic programming or similar techniques to select optimal adaptations at run-time. These techniques suffer from the "curse of dimensionality", increasing the cost of run-time adaptation decisions. We propose a novel approach that improves upon the state-of-the-art proactive self-adaptation techniques to reduce the number of possible adaptations that need be considered for each run-time adaptation decision. The approach, realized in a tool called Thallium, employs a combination of automated formal modeling techniques to (i) analyze a structural model of the system showing which configurations are reachable from other configurations and (ii) compute the utility that can be generated by the optimal adaptation over a bounded horizon in both the best- and worst-case scenarios. It then constructs triangular possibility values using those optimized bounds to automatically compare adjacent adaptations for each configuration, keeping only the alternatives with the best range of potential results. The experimental results corroborate Thallium's ability to significantly reduce the number of states that need to be considered with each adaptation decision, freeing up vital resources at run-time.

KEYWORDS

formal methods; self-adaptive systems; run-time adaptation; multi-objective optimization

ACM Reference Format:
Clay Stevens and Hamid Bagheri. 2020. Reducing Run-Time Adaptation Space via Analysis of Possible Utility Bounds. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380365

1 INTRODUCTION

Self-adaptive systems are becoming more pervasive, particularly in applications such as autonomous vehicles and medical or IoT devices [19, 24, 36, 57]. These systems need to quickly adapt to an uncertain, dynamic environment without external intervention, which is especially challenging given the nearly infinite situations such environments may present, the short window of time available to adapt, and the potentially limited computing resources available for making adaptation decisions at run-time.

In an ideal scenario where adaptations are instantaneous and immediately beneficial, a *reactive* self-adaptive system can respond effectively after a change in the environment has been detected. However, in cases such as provisioning a new cloud-based virtual machine [40], the adaptations enacted by the system may take some time, requiring a *proactive* approach which can account for the *latency* of adaptation tactics [51]. While such proactive, latency-aware (PLA) approaches promise to improve the overall fitness of the adaptations chosen [15, 45], they need to look ahead and predict future states of the environment. Recent approaches to PLA self-adaptive systems model the environment as a stochastic process independent of the state of the system [45]. Adaptation decisions can then be made via stochastic planning using the predictions of future states of the environment as input.

Historically, stochastic planning problems have been described and modeled using *Markov decision processes* (MDPs) [53], which can be solved using *dynamic programming* to optimize some utility or cost. If the number of distinct properties or settings available to the system to adapt is large, this solution can suffer from Bellman's *curse of dimensionality* [11], where the number of states that must be visited and evaluated grows factorially based on the number of values that can be assumed by the variables representing the system state. Prior decision theoretic planning research has explored ways to curtail this *state explosion*, such as minimization of MDPs [31, 46], reachability analysis [12], and machine learning [25, 54].

However, despite significant progress, the *adaptation space*—the possible state transitions that must be pondered for each run-time adaptation decision—is still enormous, even considering only allowable adaptations. This, in turn, renders dynamic run-time adaptation for real-world systems expensive in practice. This is especially problematic in volatile environments like distributed pervasive systems, where high volumes of routinely volatile software components often exist and coordinate in tandem. There is, thus, a need for methods to facilitate efficient analysis of huge adaptation spaces.

In this paper, we present a novel approach, dubbed Thallium[1], that automatically trims the adaptation space to be explored at run-time by the underlying adaptation decision maker. Unlike all prior techniques, Thallium retains only the adaptations with (Pareto-) optimal potential to provide the best utility. Thallium recognizes opportunities for trimming the dynamic adaptation space by combining the state-of-the-art structural and behavioral modeling

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380365

[1] Thallium (Tl, atomic number 81) is a metallic element with very few strong lines in its emission spectrum; the light produced by burning it is trimmed to only a few bands when passed through a prism.

techniques [16, 46] for reachability analysis with a *possibilistic* process [27] for bound analysis of the utility achievable from each system state.

Specifically, using lightweight model checking, Thallium analyzes the structural system model to produce a Markov decision process capturing the possible adaptations from each system state. It then employs probabilistic model checking to derive information from the behavioral system model about the bounds on the utility achievable from each state. Thallium then utilizes *possibilistic analysis* to resolve values representing the *potential* of each system state with respect to each optimization objective, accounting for both the positive and negative consequences of uncertainty in the environment. Such automatically derived information is used in concert with the reachability information to spot the Pareto-frontier in terms of potential for the valid adaptions from each state. Pruning away non-optimal or strictly dominated adaptations results in a trimmed MDP that only includes the adaptations that lead to states with the Pareto-optimal potential, which in turn drastically reduces the analysis time for each dynamic adaptation decision. To summarize, this paper contributes:

- *Effective adaptation space reduction at run-time.* We introduce a novel approach for effective trimming of the adaptation space to be analyzed for each run-time adaptation decision, achieving remarkable speed-ups in dynamic adaptation. The novelty of our approach comes in using a possibilistic analysis to combine prior approaches in reachability analysis with an analysis of the best- and worst-case bounds on adapatation utility.
- *Implementation:* We have realized the ideas in Thallium, a framework which relies on the Alloy lightweight model-checker [34] to generate an MDP representing the allowable adaptations of the system and on the PRISM-Games extension of the PRISM Model Checker [17, 37] to perform the bound analysis. We make Thallium publicly available to the research and education community [60].
- *Experimental evaluation*: We present results from experiments run on diverse subject systems—rigorously replicating prominent earlier studies—corroborating Thallium's ability to significantly reduce the effort required for run-time adaptation without sacrificing overall system utility.

In the remainder of this paper, we describe the motivation and intuition behind our approach in Section 2, followed by a detailed explanation of our approach in Section 3. Sections 4 and 5 cover our experiments and their results, and Section 6 discusses possible threats to validity. Sections 7 and 8 conclude with an overview of related work followed by a summary of our contributions.

2 ILLUSTRATIVE EXAMPLE

In this section, we describe the motivation and underlying insight of our research by way of an illustrative example. We consider a publish-subscribe messaging system used to monitor audio data collected from various *sources*, where each received chunk of data must be streamed to (and acknowledged by) an arbitrary number of *subscribers*. Audio data is collected from each source by a dedicated *publisher* service, which streams the data to a *relay* service for distribution to the subscribers. An overview of the relationships of the various parts of the system is shown in Figure 1.

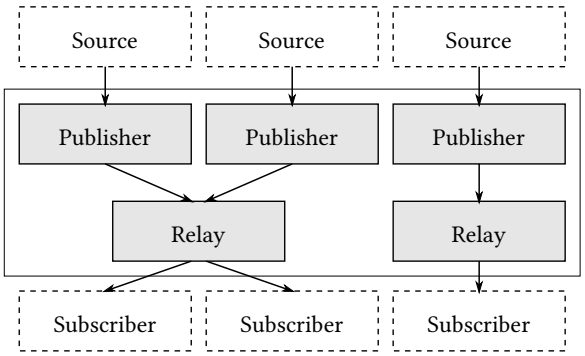


Figure 1: Pub-sub system example

Each publisher can only harvest from one source, and can only send data to a single relay. Each relay can only sustain a limited number of total connections (or *load*) comprising publishers, subscribers, and "downstream" relays, beyond which the latency of packet delivery through that relay increases sharply. The goal of the system is to deliver the streamed audio to each subscriber with as low a latency as possible, with steep penalties associated with audio latencies above a certain threshold. The system adapts to changes in its environment in the following ways:

- **PUB+**: As needed, the system can provision a new publisher service to harvest the audio from a new or otherwise unharvested source, or assign that source to a running but idle publisher. There is a cost for each online source that is not being harvested, and a cost associated with each running publisher. Provisioning a new publisher takes a non-zero amount of time before the publisher starts harvesting audio from a source, adding a tactic latency (L_p) to this adaptation.
- **RLY+**: As needed, the system can provision a new relay service for use in future connections. Each running relay service carries an attended cost for the duration of its operation, whether it is serving any connections or not. There is also some tactic latency (L_r) involved in provisioning a new relay. Each relay also has a *threshold* number of active connections, below which the relay is able to immediately handle data on that connection, and above which all connections to that relay begin to suffer additional latency (as the relay begins to buffer and switch between connections).
- **PUB-, RLY-**: As needed, the system can also deactivate a publisher or a relay (or both). No tactic latency is assumed here.
- **WAIT**: For the sake of completeness and to represent the passage of time in the case of tactics with latency, the system can elect to do nothing and simply wait for the next time step.

Figure 2 shows an example MDP that represents this system. In this Markov decision process, the states are labeled using a four-valued tuple consisting of the number of publishers, the number of relays, the number of time steps remaining for a PUB+ tactic, and the number of time steps remaining for a RLY+ tactic, respectively.

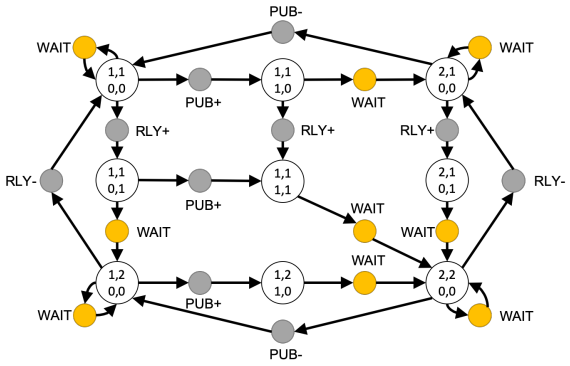


Figure 2: System MDP example (1-2 publishers/relays, latency = 1) considering reachability only. Each adaptation decision must evaluate on the order of 4 adaptations.

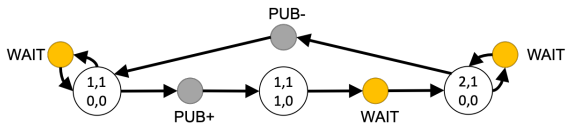


Figure 3: Trimmed MDP produced using Thallium ($T = 1000, S(e_t) \in \{0..50\}, Q(e_t) \in \{0..10\}$). Adding a new relay is not Pareto-optimal given the bounds of the environment states, and can be disregarded.

The number of publishers and of relays are both constrained to be either one or two, and $L_p = L_r = 1$.

The adaptation goal in the system is to *minimize* the total cost of the system over time, which can be computed for each time step as a function of the state of the system and the state of the environment at that time. The system state s at time t can be modeled as a tuple:

- $P(s_t)$: the number of active/ready publishers,
- $P'(s_t)$: the number of provisioned publishers (including those still starting up but not yet active/ready),
- $R(s_t)$: the number of active relays, and
- $R'(s_t)$: the number of provisioned relays.

The environment state e at time t can be modeled as a pair:

- $Q(e_t)$: the number of active sources, and
- $S(e_t)$: the number of active subscribers.

At each time step, the overall cost for that step comprises four objectives that each map the system and environment states at that time to a real value, where S is the set of all system states and E is the set of all environment states:

- The *audio latency cost*, $C_L : \mathbb{Q} \times \mathbb{N} \rightarrow \mathbb{R}$, a function of the current average load on each active relay ($\lambda : S \times E \rightarrow \mathbb{Q}$) and the connection threshold of the relays ($T \in \mathbb{N}$), where

$$\lambda(s_t, e_t) = \frac{S(e_t) + P(s_t)}{R(s_t)} + (R(s_t) - 1),$$

- The *publisher cost*, $C_p : \mathbb{N} \rightarrow \mathbb{R}$, which is a function of the number of provisioned publishers, $P'(s_t)$,
- The *relay cost*, $C_r : \mathbb{N} \rightarrow \mathbb{R}$, which is a function of the number of provisioned relays, $R'(s_t)$, and
- The *unharvested source cost*, $C_s : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$, which is a function of the number of active publishers, $P(s_t)$, and the number of active sources, $Q(e_t)$.

To make the adaptation decision, a dynamic programming solution would need to recursively evaluate the total cost that could be produced by starting from each relevant system configuration up to some finite look-ahead horizon, H, and select the adaptations leading to the state with the optimal result. H becomes yet another multiplier on the amount of work needed. The (potentially nearly infinite) values from the environment are additional multipliers, drastically increasing the number of possible adaptations. To extend our pubsub example, consider such a system where $P(s_t) \in 1, 2$, $R(s_t) \in 1, 2$, and $L_p = L_r = 1$. The configuration space allows for 16 unique configurations. If there will be at most 50 subscribers ($S(s_t) \in 0..50$) and at most 10 active audio sources ($Q(s_t) \in 0..10$) the space for the system *and* the environment would have 8,976 members. If $H = 5$—quite small—a dynamic programming solution must still find optimal adaptations among 44,880 states.

State-of-the-art systems, such as PLA-SDP [45, 46], exploit the insight that only a subset of states are *reachable* via adaptation from any given state at a given time. Some adaptations–such as adding and removing a publisher–may be mutually exclusive. Others–such as adding a relay or publisher–may take a non-trivial amount of time to complete. To utilize the additional reachability information, PLA-SDP uses lightweight model checking to exhaustively explore the structural relationships between states and construct a Markov decision process (MDP) representing the allowable adaptations within the system. In our example, it would produce an MDP like the one shown in Figure 2. This reduces the number of adaptations that need to be considered at each step within the horizon down to 4 or fewer, depending on the current state.

However, PLA-SDP may still recursively consider adaptations that would not improve the overall outcome. If each relay in the system can simultaneously manage 1000 connections (i.e., the relay connection threshold $T = 1000$), a single relay could easily handle all 50 subscribers and all 10 audio sources. Adding a second relay would only increase the cost for the number of active relays, C_r, without reducing the latency cost C_L. Given that the tactic of adding a new relay would still be *reachable*, it would still be considered by PLA-SDP for each step within the horizon, even though there would be no benefit to selecting such a strategy under any circumstances.

Thallium aims to reduce the number of sub-optimal adaptations considered by employing information about the bounds of the possible values for each objective. In the example provided above, we would consider the *best-case* to be when there are no active subscribers or sources and the *worst-case* to be when there are 50 active subscribers and 10 active sources. In either case, adding a new relay would only increase C_r without improving the cost for any other objective (i.e., it would not be Pareto optimal), so Thallium would remove those adaptations from the consideration set. With our assumed values, the MDP produced by Thallium would trim all states with more than one relay, greatly reducing the adaptation space. The final MDP for our example is shown in Figure 3.

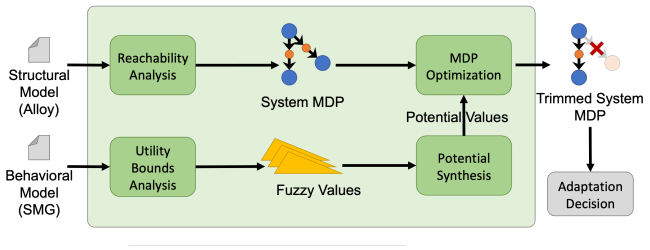


Figure 4: THALLIUM Overview

3 APPROACH

This section presents our approach to effectively trim the adaptation space at run-time. As depicted in Figure 4, THALLIUM comprises four automated components. (1) The *Reachability Analysis* component performs lightweight model checking of a provided system structural model to construct an intermediate Markov decision process (MDP) representing the states reachable via adaptation from each system configuration. (2) The *Utility Bounds Analysis* component leverages probabilistic model checking to analyze the behavioral system specification formalized as a stochastic multiplayer game (SMG). This step automatically determines new, previously unused, information about guaranteed bounds of achievable utility in the best- and worst-case scenarios, which is, in turn, used to define a fuzzy set [68] of possible utility values for each optimization objective at each state. (3) The *Potential Synthesis* component leverages possibilistic analysis [27, 67] to transform the derived fuzzy sets into a single value representing the *potential* of each state to provide the highest utility for each objective. Finally, (4) the *MDP Optimization* component compares the available adaptations at each node in the derived MDP apropos their potential for each objective and prunes strictly dominated adaptations from the final MDP.

THALLIUM requires the user to specify two formal models of the system: a *structural model* defining the states the system can assume and the transitions between them; and a *behavioral model* specifying the behavior of the system as an SMG, including the probabilities of each transition for players representing the system and its operating environment. In this paper, we present examples of the structural models using Alloy [34], which allows users to model systems using a lightweight, approachable syntax familiar to many developers. We present the behavioral models as SMGs defined using the PRISM [37] model checker, as it has been widely used in research [1, 13, 22, 23] and has extensive documentation [52]. These widely-used, familiar modeling techniques ease the burden placed on users to develop the models required for input.

3.1 Reachability Analysis

To construct an intermediate MDP representation of the system, THALLIUM generates the graph of all "reachable" states in the system using the method described by Moreno, et al. [46]. Reachability is defined using the following predicates for system states $c, c' \in C$:

- *Immediate Transitions* $R^I(c_0, c_1) - c_1$ can be reached from c_0 using an immediate transition (i.e., applying no tactic or one with no latency);
- *Delayed Transitions* $R^D(c_0, c_1) - c_1$ can be reached from c_0 in one time interval using a delayed transition by applying a tactic with latency; and
- *Transitive Transitions* $R^T(c_0, c_n) - c_n$ can be reached in one time interval using a delayed transition followed by one or more immediate transitions, or more formally:

$$\exists\, c_1, ..., c_{n-1} : R^D(c_0, c_1) \wedge R^I(c_1, c_2) \wedge ... \wedge R^I(c_{n-1}, c_n)$$

These predicates can be used to fully define the feasible transition matrix of the system configuration MDP, which requires evaluating a large number of possible combinations of tactics/transitions. To generate such a transition matrix, we use Alloy [34] to formally specify both the system configuration and the reachability predicates. Alloy is a formal modeling language based on relational logic, amenable to fully automated yet bounded analysis. It facilitates representing abstract system structures and the relationships between them as a set of constraints. Once a system has been described as a collection of structural type signatures and constraints, the Alloy Analyzer can be used to automatically find model instances that satisfy all the constraints.

Three Alloy specifications are conjoined to define the structural system configuration model: (1) the system state specification, (2) the tactics specification that can execute transitions between states, and (3) a trace specification that maps a configuration state to a set of other configuration states that can be reached by sequential applications of allowed tactics. Figure 5 partially represents the system state specification for our running example (cf. Section 2). According to lines 19–23, the `SystemCfg` signature contains two fields, `activeRlys` and `activePubs`, that represent the number of active relays and publishers, respectively; the `prog` field represents the latency of any delayed tactics as a unique progress value for each tactic with latency.

The next step is to formalize the progress on each particular tactic, which captures whether the tactic will reach a state succeeding the current state in the model's ordering. Such formal specification of each tactic's progress enables us to automatically compute R^D. Each tactic will need a predicate definition to check whether the successor thereof can be reached in a single time interval and to model the effect of the tactic completion. As a concrete example, Figure 6 represents the `AddRly_Prog` tactic progress specification. This predicate holds between two states for transitions involving the `AddRly` tactic with latency.

Given a concrete analysis scope that bounds the search for each top-level signature in the system specification, e.g., `SystemCfg` and `Prog`, THALLIUM uses the Alloy analysis engine to exhaustively generate all model instances that satisfy the R^D predicate. Those instances are then encoded in a lookup table for use at run-time.

Computing R^I is conducted by automatically generating traces representing the sequential applications of each no-latency tactic that can be applied to transitions from one state to another. Figure 7 represents the trace signature definition along with its predicates for the `AddRly` no-latency tactic. According to lines 4–6, the `TE` trace signature contains two fields–`config` and `start_tactics`–that represent a particular system configuration and the set of tactics needed to be started to reach that state from the preceding state in the trace, respectively. Tactic-specific predicates formalize both

```
sig RlyCount {} // each value in the system state needs a
sig PubCount {} // signature to represent its domain

abstract sig Tactic {} // tactics
abstract sig LTactic   // tactics w/ latency
  extends Tactic {}

// define all tactics with latency
one sig AddRly,  AddPub  extends LTactic {}
// define all instantaneous tactics
one sig DropRly, DropPub extends Tactic  {}

// each tactic with latency requires a progress value
abstract sig Prog {}
sig AddRly_P extends Prog {}
sig AddPub_P extends Prog {}

// the system configuration itself
sig SystemCfg {
  activeRlys : one RlyCount,
  activePubs : one PubCount,
  prog       : LTactic -> Prog
} {
  // every tactic in LTactic has a progress...
  prog.univ = LTactic
  // ...at most one...
  ~prog.prog in iden
  // ...and each from its own domain.
  prog[AddRly] in AddRly_P
  prog[AddPub] in AddPub_P
}
```

Figure 5: System state specification for the pub-sub example (cf. Section 2), including instantaneous tactics (`DropRly`, `DropPub`) and tactics with latency (`AddRly`, `AddPub`). Latency is modeled by adding a progress field to the configuration for each tactic with latency.

```
open util/ordering[RlyCount] as RlyCount_O
open util/ordering[AddRly_P] as AddRly_PO

pred AddRly_Prog[c,c' : SystemConfig] {
  (c.prog[AddRly] ≠ AddRly_PO/last) implies {
    // if the tactic is running, the predicate holds for
    // successor states that have the next progress value
    c'.prog[AddRly] = AddRly_PO/next[c.prog[AddRly]]
    c'.prog[AddRly] = AddRly_PO/last implies {
      // if that is the last progress value, then
      // it holds if the relevant state value has been
      // updated to reflect the tactic's execution
      c'.activeRlys = RlyCount_O/next[c.activeRlys]
    } else {
      // otherwise, the state value stays the same
      c'.activeRlys = c.activeRlys }
  } else {
    // if the tactic is not running, assert that it
    // stays the same in the successor
    c'.prog[AddRly] = AddRly_PO/last
    c'.activeRlys = c.activeRlys }}
// the progress predicates can be composed sequentially
// to compute the actual delayed reachability predicate
pred RD[c,c' : SystemConfig] {
  some tc : SystemConfig |
    AddRly_Prog[c, tc] and AddPub_Prog[tc, c'] }
```

Figure 6: Tactic progress predicate for the pubsub example. This predicate holds between two states for transitions involving a tactic (`AddRly`) with latency.

whether a particular tactic can be applied to the current system configuration (lines 9–13) and the results of applying an instantaneous configuration if applicable (lines 15–24). The `traces` fact (lines 25–32) specifies which transitions from one system configuration to another are valid by composing the applications of the various

```
open util/ordering[TE] as trace

// Trace of sequential applications of no-latency tactics
sig TE {
  config : one SystemCfg
  start_tactics : set T }
// Checking whether the tactic can be applied to the
// current system configuration
pred AddRly_ok[e : TE] {
  // tactic not running
  e.config.prog[AddRly]] = AddRly_PO/last
  !(AddRly  in e.start_tactics)
  !(DropRly in e.start_tactics) }
// Initiate the progress for the AddRly tactic
pred AddRly_start[e, e' : TE] {
  AddRly_ok[e]
  e.config.activeRlys ≠ RlyCount_O/last
  e'.start_tactics = e.start_tactics + AddRly
  let c = e.config, c' = e'.config | {
    // start the progress for this tactic,
    // and change nothing else
    c'.prog[AddRly] = AddRly_PO/first
    equals[c, c']
    (LTactic - AddRly) <: c.prog in c'.prog }}
fact traces {
   let fst = trace/first |  fst.starts = none
   all e : TE - trace/last | let e' = next[e] | {
      equals[e, e'] and equals[e', trace/last]
   } or DropRly_enact[e, e']
     or DropPub_enact[e, e']
     or AddRly_start[e, e']
     or AddPub_start[e, e'] }
```

Figure 7: Example trace predicates to track pubsub system evolution. The `traces` fact defines which transitions from one configuration to another are valid by composing the application of the tactic-specific predicates.

tactic-specific predicates, e.g., `DropRly_enact` and `AddRly_start`. Thallium again uses the Alloy analysis engine to obtain all valid traces (i.e., those satisfying the `traces` fact) and capture them in a lookup table for R^I to be used at run-time.

3.2 Utility Bounds Analysis

To compute the upper- and lower-bounds on the overall system utility achievable from a given state, Thallium analyzes the input behavioral model as a *stochastic multiplayer game* (SMG), in which players alternate selecting transitions based on the transitions' assigned probabilities. The SMG is analyzed using a probabilistic model checker [17] as described by Cámara, et al. [16].

The system configuration and its transitions modeled in the SMG, are controlled by one *player*, and the environment's state is controlled by another player. The players take turns selecting from the available transitions in the underlying model until the time horizon, measured in discrete time steps, has been reached. Depending on the properties set on the execution of the game, the players can either cooperate to achieve a shared goal or compete by pursuing their own individual goals. In the context of PRISM-Games, these properties can be expressed using the rPATL logic [17], a branching time temporal logic based ultimately on ATL[2]. It has been widely used for game-theoretic reasoning systems involving multiple agents, such as the system and environment in adaptation decisions.

Our research uses both the coalition operator $\langle\langle C\rangle\rangle$ adopted from ATL and the reward operator $\langle\langle C\rangle\rangle \mathrm{R}^{r}_{max=?}[\mathrm{F}^{*}\phi]$, which is an extension of the generalized reward operator, R [28]. This operator

quantifies the maximum accumulated reward r that can be guaranteed by the players in coalition C along any paths leading to a state satisfying ϕ, regardless of the actions taken by any players outside the coalition. We use PRISM-Games to verify two properties of the *upper-bound for the cumulative reward* and the *lower-bound for the cumulative reward* for each reachable system configuration. In the following formulas, *sys* is the player representing the adaptive system, *env* is the player representing the stochastic environment, and ω is a predicate defining a termination state.

$$\langle\langle sys \rangle\rangle \mathrm{R}^{U}_{max=?}[\mathrm{F}^{c}\,\omega] \tag{1}$$

$$\langle\langle sys, env \rangle\rangle \mathrm{R}^{U}_{max=?}[\mathrm{F}^{c}\,\omega] \tag{2}$$

Property 1 (the *lower-bound for the cumulative reward*) represents the maximum utility U for the overall system, that can be *guaranteed* by *sys* alone along any path leading to a terminal state; this is the *worst-case* that can be achieved by an optimal player. Property 2 (the *upper-bound for the cumulative reward*) represents the *best-case* for the overall system utility, wherein *sys* and *env* are working together to maximize the reward over time.

The PRISM-Games commences with a definition of both the players and the rewards, based on the relevant utility functions, followed by a series of modules for each set of independent, i.e., not mutually-exclusive, tactics and the environment evolution. The turns are controlled by a specific module for each turn, which is synchronized with the tactic modules for that turn. The time intervals and horizon are modeled using a special *clock* module that keeps track of the discrete time progression. Rewards are assigned during a distinct *reward* turn by a corresponding module.

As a concrete example, Figure 8 shows a tactic module from the pubsub behavioral model of the pubsub system. Each tactic module includes all possible outcomes for the MDP. For example, five possible outcomes for the relay module tactic have been modeled in the figure, where each one is labeled using the same transition label. This allows PRISM-Games to resolve each step in the simulation using whichever of the concurrent tactics would provide the best strategy for the coalition.

Similarly, each possible evolution of the environment's state must be provided as a possible choice of action for PRISM-Games, which we define in a module dedicated to that evolution. Figure 9 shows an example environment evolution module. It represents the number of active subscribers in the pubsub system. A similar module would be synchronized using `do_env` to simulate the number of audio sources in the pubsub environment.

We can then resolve the lower- and upper-bounds for the cumulative reward, as shown in Equation 1 and Equation 2, respectively. We vary the initial system state values (e.g., `INIT_RELAYS` and `INIT_RPROG`) for each tactic in order to determine the worst- and best-case utility that can be attained over the time horizon, starting from each admissible system configuration. The admissible system configurations include configurations where the progress on a latency-bearing tactic is non-zero, indicating that the system had started, but not yet finished, executing a tactic.

Running the PRISM-Games models described above for each example computes a value for the properties defined in Equations 1 and 2. We vary the constants used in the model to simulate starting from each possible system configuration. This produces a best-

```
module relay
  relays     : [MIN_RELAYS..MAX_RELAYS]         init INIT_RELAYS;
  relay_prog : [MIN_RELAY_PROG..MAX_RELAY_PROG] init INIT_RPROG;
  // skip; do nothing
  [do_sys] (turn=SYS_TURN) ->
    true;
  // initiate the latency-bearing tactic to inc. the value
  [do_sys] (turn=SYS_TURN) & (relays<MAX_RELAYS)
      & (relay_prog=MIN_RELAY_PROG) ->
    (relay_prog'=MAX_RELAY_PROG);
  // advance the progress for the latency-bearing tactic
  [do_sys] (turn=SYS_TURN)
      & (relay_prog>(MIN_RELAY_PROG+1)) ->
    (relay_prog'=(relay_prog-1));
  // finish the latency-bearing tactic
  [do_sys] (turn=SYS_TURN)
      & (relay_prog=(MIN_RELAY_PROG+1)) ->
    (relay_prog'=MIN_RELAY_PROG) & (relays'=(relays+1));
  // enact the no-latency tactic to decrement the value
  [do_sys] (turn=SYS_TURN) & (relays>MIN_RELAY_VAL)
      & (relay_prog=MIN_RELAY_PROG ->
    (relays'=(relays-1));
endmodule
```

Figure 8: Example tactic module from the pubsub behavioral model. The `relay` module has both an instantaneous tactic (to drop a relay) and a tactic with latency (add a relay) and must keep track of tactic progress.

```
module subscriber
  subscribers : [MIN_SUBS..MAX_SUBS] init INIT_SUBS;
  [do_env] (turn=ENV_TURN) -> (subscribers'=MIN_SUBS);
  [do_env] (turn=ENV_TURN) -> (subscribers'=(MIN_SUBS+1));
  // ...
  [do_env] (turn=ENV_TURN) -> (subscribers'=MAX_SUBS);
endmodule
```

Figure 9: Example environment evolution module representing the number of active subscribers in the pubsub system.

and worst-case value for the cumulative utility achievable from that configuration. We can also compute the expected value using PRISM by simulating the expected, probabilistic reward (using the **R** operator) with no coalition of players.

Those three values are interpreted to define a triangular distribution of a fuzzy value, representing the membership of a utility value in the set of possible utility values for the given reward. The worst-case value (U^p) represents the *most-pessimistic* result possible; no worse outcome is achievable so the degree of membership in the set of possible values is taken to be zero. Similarly, the best-case value (U^o) is assigned a degree of membership of zero, since it is the *most-optimistic* value possible. The expected value ($E(U)$) is assigned a degree of membership of one; it is the *most-likely* value, and therefore should be a member of the set.

3.3 Potential Synthesis

The fuzzy values generated by computing the utility bounds are then normalized and optimized to produce one value per reward per state representing the "*potential*" of that state with respect to that reward. The key insight in computing the potential state value per reward, inspired by POISED [27], is to simultaneously optimize three values from the possibility distribution by selecting a system configuration. We define the optimization objectives as follows:

(1) Maximize the expected reward (or minimize the expected cost), $z_e \equiv E(U)$,

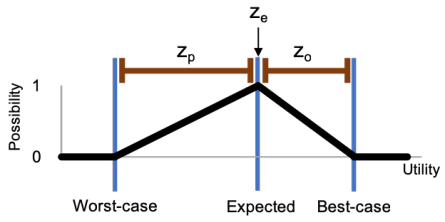


Figure 10: Triangular possibility distribution with optimization values (z_p,z_e,z_o) indicated. z_e is the expected or most-likely value, z_p the "down-side", and z_o the "up-side".

(2) Maximize the positive consequences of uncertainty or "upside" potential, $z_o \equiv |U^o - E(U)|$, and
(3) Minimize the negative consequences of uncertainty or "downside" potential, $z_p \equiv |U^p - E(U)|$.

Figure 10 depicts a schematic representation of the quantities involved in the calculation of the potential state value along with a triangular possibility distribution. In order to compare two of these fuzzy sets we must be able to resolve any trade-offs between the values. For example, for one configuration, the value of U_o or "upside" may be greater than that of an alternate configuration (which is desirable), but the value of U_p or "downside" may also be greater (which is not desirable). Also, the ranges for each of the three objective values may differ, making direct comparison more difficult. Therefore, we first normalize the values, and then reframe the overall problem as a single-objective optimization problem.

The values can be normalized by applying a linear normalization function, μ_{z_j}, to map the domain of each of the three objectives z_j (where $j \in p, e, o$) to a value between 0 and 1. A mapped value of 0 corresponds to the minimum value of z_j and 1 corresponds to the maximum value, with a linear mapping of all intervening values [38]. With the values normalized, we reframe the multi-objective optimization problem as a single-objective optimization function via optimizing an auxiliary function ψ, returning the weighted minimum of the normalized values, as shown in Equation 3 with w_j representing the weight :

$$\underset{c \in C}{\text{argmax}}\ \psi \text{ where } \psi \leq w_j \mu_{z_j} \text{ and } \sum_j w_j = 1 \text{ for } j \in \{p, e, o\} \quad (3)$$

By maximizing the auxiliary function we also optimize the original objectives. Thallium stores the maximized value of ψ for each system configuration as the *potential* metric for each configuration and uses it in the next step for trimming the MDP transitions.

3.4 MDP Optimization

With the potential values determined for each configuration and each reward, Thallium can then compare adjacent transitions to determine which transitions, if any, can be eliminated from the generated reachability graph (cf. Section 3.1).

Thallium compares each configuration by finding which of the neighboring transitions will move the system into a state whose potential is Pareto optimal with respect to all of the reward objectives, rather than using a single weighted composite value representing all of the objectives. More formally, a transition t to a system configuration c will only be retained if there is no other transition t' to a different configuration c' such that the potential of any of the rewards at c' is greater than the same reward potential at c *and* the potential of all other rewards at c' is at least equal to the corresponding potential at c. All other transitions will be trimmed, leaving only the Pareto-optimal siblings to be considered when selecting adaptations. An example using the pubsub system from Section 2 is shown in Figure 11, which shows potential values from starting state (1,1,0,0). The (1,1,0,1) and (1,1,1,1) states are dominated by the others for every objective, so adaptations to those states (i.e., adding a relay) can be trimmed. The (1,1,0,0) and (1,1,1,0) states form the Pareto frontier, since the former is better for C_p and the latter for C_s while they are equal for C_L and C_r.

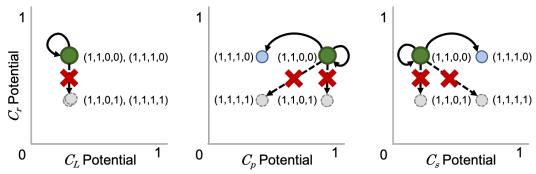


Figure 11: Example potential values for pubsub system, starting from (1,1,0,0). (1,1,0,1) and (1,1,1,1) are dominated by the others for every objective, so the transitions to those states (i.e., adding a relay) can be trimmed. (1,1,0,0) and (1,1,1,0) form the Pareto frontier, as the former is better for C_p and the latter for C_s while they are equal for C_L and C_r.

The resulting trimmed MDP produced by Thallium is then used at run-time to facilitate efficient adaptation decision making, without sacrificing the overall system utility.

4 EVALUATION

Our evaluation of Thallium addresses these questions:

- **RQ1**: How well does Thallium perform in reducing the adaptation space both offline and at run-time?
- **RQ2**: Does Thallium provide comparable utility for the system compared to the state-of-the-art approaches?

To answer these questions, we have designed and conducted experiments using the apparatus we developed based on the presented approach. Thallium uses the Alloy Analyzer [34] version 4.2 to model the system configuration and generate the system MDP, and the PRISM-Games [17] extension of the PRISM Model Checker [37] version 4.3 to determine the utilities of the best, worst, and most-possible cases. We developed Thallium as a custom Java program which generates possibility values and trims sub-optimal adaptations from the system configuration MDP.

4.1 Experimental Subjects

We used two exemplar systems—each from a separate domain—for our experiments: (1) SWIM [48], a simulator of self-adaptive web infrastructure; and (2) DART [33, 43], a system that simulates a team of unmanned aerial vehicles (UAVs) detecting targets while avoiding threats during a reconnaissance mission. These systems are described in more detail below.

SWIM: Our first experimental subject is SWIM (Simulator of Web Infrastructure and Management) [43], a self-adaptive system designed to simulate a load-balanced web application. The system supports adding or removing servers as well as controlling a dimmer setting that regulates the level of content returned for each request. SWIM is implemented on top of OMNeT++, a discrete event simulation environment [63], and simulates only the high-level processing of web requests as computational work. The system is highly extensible, providing monitoring information about the current state of the simulation and effectors to simulate the execution of tactics.

We define SWIM's system configuration, $c_i \in C$, during an arbitrary discrete time step, i, as the tuple (s_i, d_i, Δ_i), where s_i represents the number of active servers, d_i represents the setting of the dimmer—which increases or decreases the percentage of optional content, with a corresponding decrease or increase to the service rate, respectively—and Δ_i represents the number of time steps remaining for an active tactic execution. The environment state, e_i, for the same time step represents the number of web requests that arrived at the server during that time step. The adaptation goals of the SWIM system are to (1) minimize the average response time, (2) minimize the server provisioning costs, and (3) maximize the percentage of optional content delivered.

DART: As the second experimental subject, we use the DART self-adaptive system developed by the Carnegie Mellon Software Engineering Institute [33, 47]. DART simulates a team of UAVs flying a reconnaissance mission in formation over a bounded, hostile environment containing both reconnaissance targets and enemy threats. One of the drones is designated as the leader, and autonomously decides the actions the team should undertake in order to fulfill the mission goals, such as changing altitude or formation. In our experimental scenario, the team flies along a planned route of D equal segments at a constant rate, using a downward-facing sensor to detect as many targets as possible while avoiding destruction from any threats. Both the number and location of the targets and threats are static, but unknown at the beginning of the run. Due to that uncertainty, the team must self-adapt by changing altitude or formation to maximize the number of targets safely detected. Flying at a lower altitude increases the probability of detecting a target, but also increases the chance of being destroyed by a threat. Similarly, flying in a loose formation provides better target detection while a tight formation decreases the probability of destruction.

4.2 Experimental Design

To address our research questions, we compared Thallium with the state-of-the-art technique in proactive, latency-aware self adaptation, namely PLA-SDP [46]. We conducted three experiments on our exemplar systems. For each, we applied both PLA-SDP and Thallium to construct MDPs for the system. The MDP generated by PLA-SDP for each system was considered a baseline for each of the three experiments, against which we compared the relevant outcomes using the trimmed MDPs produced by Thallium. All three experiments were run on a MacBook Pro with a 2.3 GHz Intel Core i5 processor and 16 GB of RAM.

Experiment 1. For the first experiment, we sought to evaluate Thallium's effectiveness at reducing the size of the static adaptation space represented by the system MDP. Both PLA-SDP and Thallium rely heavily upon the modeling choices and utility functions specified with the structural and behavioral models provided as input, but Thallium also introduces additional configuration–the weights assigned to the optimization objectives during potential synthesis (see Section 3.3). As such, we used Thallium to generate multiple MDPs for each system, with different weights for each. We compared the number of transitions in each of the those resulting MDPs to the number of transitions in the baseline MDP, reporting the outcome as a percentage reduction compared to the baseline.

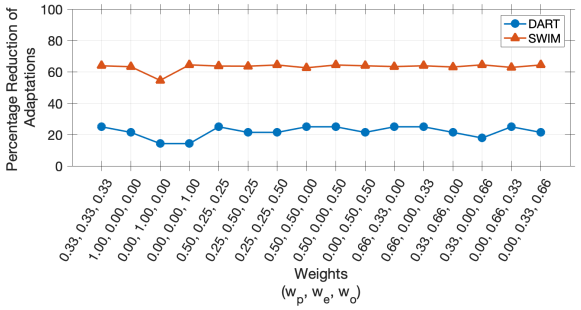
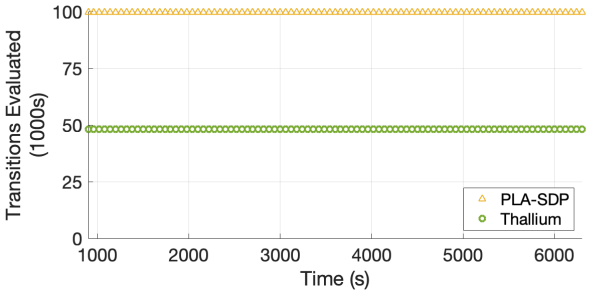


Figure 12: Percentage of adaptations trimmed by Thallium from baseline (PLA-SDP) MDP for each evaluated system using different values for (w_p, w_e, w_o) (see Section 3.3)

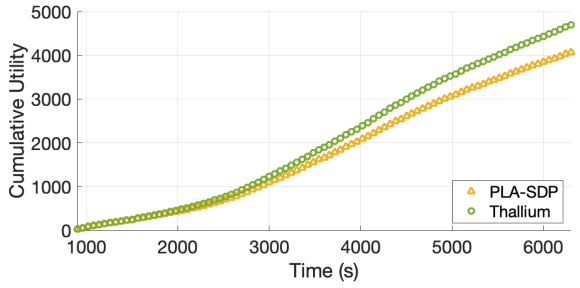
Experiments 2 and 3. For the second and third experiments, we executed simulators included with the SWIM and DART exemplars, respectively. We tested (1) the number of adaptations dynamically evaluated at run-time and (2) the overall system utility achieved when adapting based on the baseline MDP vs. a trimmed MDP generated by Thallium. We varied the adaptation manager provided to guide adaptation–either the MDP from PLA-SDP or Thallium–and measured the number of adaptations evaluated during each run-time adaptation decision and the overall utility measurement relevant to each exemplar. In addition, we introduced a purely reactive adaptation manager to serve as a third comparison for Experiment 2, detailed in Section 4.4.

4.3 Experiment 1: Static MDP Trimming

For our first experiment, we evaluated Thallium's effectiveness in reducing the size of the static adaptation space compared to PLA-SDP for each of our exemplar systems. As discussed in Section 3.3, the potential value generated for each system state and used for the Pareto optimization is synthesized by normalizing and optimizing the three values from a triangular possibility distribution—z_p, z_e, z_o—each with a corresponding weight. Since the weighting needs to be specified by the user, we also sought to quantify the impact of the weighting on the reduction of the adaptation space. Therefore, we ran Thallium on each exemplar system using each of sixteen valid weight assignments, ranging from equal weighting to exclusively considering a single value. Figure 12 summarizes the results for each system and each weighting, reporting the percentage of allowable adaptations trimmed by Thallium compared to the number in the MDP generated by PLA-SDP.



(a) Num. adaptations evaluated for each adaptation decision



(b) Cumulative system utility

Figure 13: Comparing Thallium and the state-of-the-art PLA-SDP technique for the SWIM self-adaptive system [48]. Thallium (a) reduces the number of adaptations considered by 51.9%, and (b) achieves higher cumulative utility.

For the SWIM exemplar, PLA-SDP generated a total of 340 possible system configurations, connected by 1,172 admissible adaptations. The model provided to the *Utility Bounds Analysis* component was drawn from a real-world trace of TCP activity published by ClarkNet [3] which fluctuated between 0 and 130 requests per period. Thallium trimmed between 640 and 756 adaptations from consideration, reducing the adaptation space by between 54.6% and 64.5% (as shown in Figure 12). Averaged across all weight settings, Thallium reduced the number of adaptations for SWIM by 63.2%. For DART, PLA-SDP devised 80 system configuration states with 112 admissible adaptations; Thallium reduced that number by an average of 21.9%, trimming between 16 and 28 adaptations.

Overall, the differing weights in all cases produced only a small variation in the number of adaptations trimmed from the baseline MDP, consistently hewing close to the mean. For SWIM, the relative standard error in the percentage of adaptations trimmed across all 16 weightings was only 1%; for DART, it was 4%. *For each weighting evaluated, Thallium consistently trimmed adaptations when compared to PLA-SDP, and all of the tests across all the subject systems resulted in a reduction of the adaptation space.*

4.4 Experiment 2: SWIM Simulation

Our second experiment evaluated Thallium's impact at run-time through the use of the SWIM network simulator. The experiment measures two values: (1) the number of adaptations evaluated as part of each adaptation decision and (2) the overall cumulative utility reported by the simulator. The results for both measurements are summarized in Figure 13.

For the first number of adaptations evaluated, we found large differences in the number of run -time comparisons performed using each approach during adaptation decisions. Figure 13(a) shows the number of joint system/environment states to which adaptation was evaluated. The adaptation manager based on PLA-SDP averaged 99,989 comparisons per adaptation decision. This number is much higher than the 1,172 adaptations present in the *system* MDP generated by PLA-SDP, as the look-ahead done by the proactive manager and the uncertainty in intertwined environmental states balloons that number in order to fully evaluate each possible adaptation at run-time. Thallium cuts that number in half, to 48,108 run-time comparisons (a reduction of 51.9%) for *each* adaptation decision.

Figure 13(b) summarizes the overall system utility determined by the SWIM simulator for PLA-SDP and Thallium. PLA-SDP generated a cumulative utility value of 4,067 over the course of the trace, while Thallium achieved a higher cumulative utility total, with a final tally of 4,692. The difference in the observed cumulative utility between PLA-SDP and Thallium becomes more and more pronounced as the execution time increases. Alongside PLA-SDP and Thallium, we also assessed a third, purely reactive adaptation mechanism [43] that responded to the observed response time (not shown). The reactive manager frequently violated the 750 millisecond response time threshold imposed by the simulator's utility function, resulting in a *negative* cumulative utility due to the accrued penalties.

4.5 Experiment 3: DART Simulation

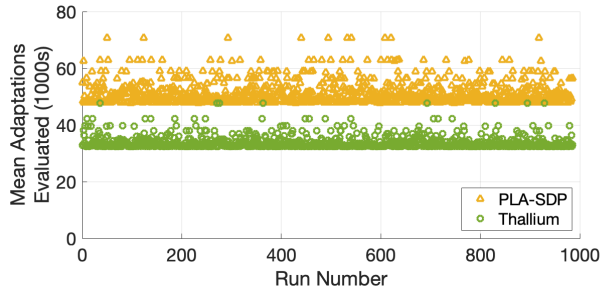
Our third experiment evaluated Thallium's impact at run-time through the use of the DARTSim simulator [47] of the DART system. The simulator plotted the course of a drone team over a randomly-generated, 40-step course containing four targets and six threats, allowing the drones to adopt two formations and one of five altitudes.

Figure 14 depicts the results of experiments conducted over 985 such courses. According to the experimental results, a remarkable reduction in the number of adaptations evaluated at run-time is observed when using Thallium. DART with PLA-SDP performed 51,385 comparisons per evaluation (on average) across the 39,400 simulated evaluation periods, whereas Thallium compared an average of only 33,886 joint states (34.1% fewer).

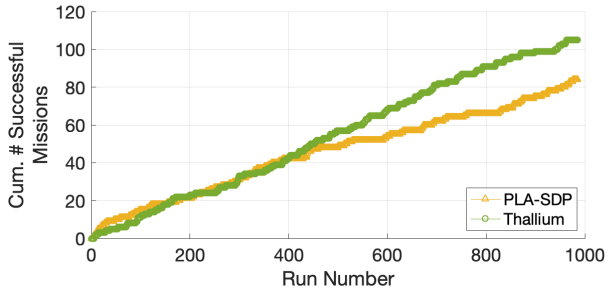
For overall utility, DART determines whether or not each mission was an overall success, i.e., the team was not destroyed *and* it detected at least half of the targets. Figure 14(b) summarizes the successful missions by both PLA-SDP and Thallium. The success rate achieved by Thallium is noticeably higher than that of PLA-SDP. Specifically, out of the 985 runs, PLA-SDP resulted in 84 successful missions total, whereas Thallium was successful in 105.

5 DISCUSSION

This section details our interpretation of the results obtained with respect to each research question and presents some limitations.



(a) Mean adaptations evaluated per simulation.



(b) Cum. num. successful simulations.

Figure 14: Comparing Thallium and the state-of-the-art PLA-SDP technique over the DART self-adaptive system. Thallium (a) reduces the number of adaptations considered by 34.1%, and (b) produces more successful simulation runs.

5.1 RQ1: Adaptation space reduction

All three experiments contributed some data to address *RQ1*, which questioned Thallium's efficacy at reducing the adaptation space compared to state-of-the-art techniques. Experiment 1 evaluated the reduction in the number of adaptations represented in the static model of each exemplar. In all cases (cf. Figure 12), Thallium successfully reduced the number of adaptations compared to PLA-SDP, with an average reduction of 63.2% for SWIM and 21.9% for DART. Furthermore, these results show that Thallium consistently reduces the adaptation space even as the weightings for the potential synthesis vary. The relative standard error for each exemplar was quite low (1% for SWIM and 4% for DART) indicating that Thallium provides a robust improvement, no matter the specific tuning of its additional input parameters. Also, as shown in Experiments 2 and 3 (cf. Figures 13(a) and 14(a), respectively), Thallium significantly cuts the number of run-time comparisons done during the adaptation decision. For SWIM, the number is cut in half (51.9%), from 99,989 to 48,108 comparisons. We observed the same trend with DART, where Thallium compared only 33,886 joint states rather than 51,385 (34.1% fewer). We interpret these data as supporting our insight that information about the bounds of possible utility can be employed to reduce the explosion of the adaptation space.

5.2 RQ2: Overall utility of adaptation strategy

Our next criterion was the overall utility achieved by each technique. Experiments 2 and 3 collected data on the utility by simulating the SWIM and DART exemplars, respectively. As shown in Figures 13(b) and 14(b), Thallium produced higher cumulative system utility (4,692) for the SWIM simulation compared to PLA-SDP (4,067), as well as more total successes among the 985 simulated missions run with the DART simulator (105 for Thallium vs. 84 for PLA-SDP). We interpret this data as suggesting that Thallium, despite significantly trimming the search space, provides better overall utility compared to the state-of-the-art techniques in self-adaptation.

5.3 Limitations

Extending our work to apply to other systems and situations is subject to a few limitations. First, Thallium is specifically targeted to reducing the state space in solutions that operate on a Markov decision process constructed to represent the structure of the system itself, independent of the stochastic environment. Systems where the two are not independent would require a modification to this approach, but the underlying insight of trimming a graph-based problem space should still apply. Second, Thallium is most suitable in situations where there is a trade-off in the choice between objectives. If one objective tends to dominate the others, there are fewer opportunities to trim the search space. Third, as described in Section 4.3, Thallium and similar formal analysis techniques—such as PLA-SDP—are heavily dependent on models of the system (both structural and behavioral) and the utility functions provided as part of the world specified by the user. The models obviously must represent the system with a high fidelity in order to provide correct results, but checking the models for correctness is beyond the scope of this paper. As an example of the impact of the utility function, the SWIM simulation used in our evaluation solved the underlying decision problem using a weighted sum of three optimization objectives: a cost based on the number of provisioned servers, a reward for having a higher dimmer value (delivering more optional content), and a reward for a low response time. Following Moreno, et al. [43, 46], those objectives were weighted as 0.4, **0.0**, and 0.6 respectively. In our evaluation, this led to both PLA-SDP and Thallium immediately reducing the dimmer to the lowest level in order to gain the benefit to the service rate. While that may seem like a surprising strategy, it makes perfect sense given the lack of a penalty. While our evaluation shows Thallium's general applicability in reducing adaptation space, specific outcomes are very sensitive to manual parameter tuning of this sort, as are other similar approaches such as PLA-SDP [43].

6 THREATS TO VALIDITY

The internal validity of these results relies on the correctness of our custom implementation of part of the approach, particularly the components used to optimize and generate the metric for each node's potential (Section 3.3) and that used to compare and trim the output MDP using Pareto analysis (Section 3.4). Conceptually, our possibilistic analysis was based directly on prior research [27, 67] and comparing multiple objectives to determine Pareto optimality is a well-known problem. Therefore, our correctness would only

be threatened due to the skill of the implementer. To address this concern, we thoroughly validated all of our tool components to ensure their correctness. Moreover, by using the same objects as our baseline systems we can compare the results produced by Thallium with those previously reported to help ensure correctness, at the potential cost of external validity. While these systems are small and exhibit only a limited number of available adaptations for each decision point, we believe that they are representative of real-world self adaptive systems (e.g., autonomous load balancers). We also believe that systems with more choices at each step would see even more benefit from our approach as they have more potential states to trim. We intend to explore larger systems in future work on this topic.

7 RELATED WORK

Many studies have utilized structural or architectural models to drive self-adaptation [20, 21, 30, 36, 61, 65, 66], including proactive self-adaptation [14, 32, 41, 42, 51, 64]. In particular, Thallium is largely founded on the research done by Moreno, et al. [43–46, 49] and Cámara, et al. [15, 16] on proactive, *latency-aware* self-adaptive systems. The PLA-SDP system used as a baseline for our evaluation was drawn from this body of work [46]. That approach pioneered the modeling of the reachable states in a self-adaptive system, which our approach improves upon. Rather than relying only on reachability, Thallium also leverages information about the bounds of possible utility to evaluate the potential of each state. This bound information is extracted from the behavioral model by simulating a stochastic multiplayer game, as described by Cámara, et al. [15, 16]. While they use the bound information simply as a verification technique, Thallium includes that additional information about the possible behavior of the system as a way to improve overall adaptation decision making.

Esfahani, et al. [27] use the mathematics of fuzzy sets [68] and possibility theory [67] to evaluate the likelihood each system state presents to satisfy the requirements of the system. In their work, they establish the bounds of the fuzzy sets using confidence intervals on the probability distribution for each uncertain element gathered from observation or stakeholder interviews. Our approach is novel in determining the bounds instead based on the guarantees from the behavioral model.

Thallium addresses the adaptation state explosion by analyzing the bounds, but there are other approaches to reducing the space. Quin, et al. [54] recently propose an approach that adds a learning module to the MAPE-K loop to select subsets of adaptations for consideration at run-time. FUSION [26] also uses learning to reduce the adaptation space, as well as Integer Programming solvers to solve the underlying optimization problem. Learning-based approaches could be employed on the reduced adaptation space from Thallium, suggesting future research in combining the approaches.

PLATO [55] and VALKYRIE [29] employ evolutionary algorithms to solve single objective optimization problems. Pascual, et al. [50] present an approach to using multi-objective evolutionary algorithms (MOEAs) to optimize configurations for dynamic software product lines. FEMOSAA [18] also utilizes MOEAs to select configurations, but improves upon previous work by giving preference to knee solutions along the Pareto frontier. Knee solutions may provide a better balance of objectives for the final solution. Identifying knee solutions while trimming transitions is a possible avenue for future research with Thallium.

8 CONCLUSION AND FUTURE WORK

In this paper, we presented Thallium, a novel technique to reduce the number of possible system configurations that need be considered for a self-adaptive system to make adaptation decisions at run-time. Thallium improves upon prior research in proactive, latency-aware self adaptation by using probabilistic model checking to glean information about the bounds of system behavior. We find a value representing each state's *potential* through possibilistic analysis, and trim all transitions to states that are strictly dominated by neighboring states across all objectives.

In the experiments conducted over the self-adaptive systems drawn from recent studies, Thallium was shown to significantly trim the transitions under consideration. The experimental results further corroborate that Thallium, despite considerably shrinking the search space, features higher overall utilities for all the systems under analysis compared to the state-of-the-art techniques.

In future research, we hope to evaluate the approach we have taken with Thallium in different domains and extend the approach to leverage new and different analysis techniques. We also hope to explore additional, related techniques to further reduce the decision space for self-adaptive systems.

Trade-off analysis [5–10, 35, 62] provides a potentially fruitful avenue for further research related to Thallium. These analysis techniques often utilize formal models to synthesize a set of possible system architectures or design choices and optimize among competing objectives to select a set of candidates to present to a user. While static analysis of the tradeoffs can often be done cheaply, a full dynamic analysis can be more expensive. The behavioral and possibilistic analysis done by Thallium could possibly be employed to eliminate some candidate systems *a priori* before performing the expensive, dynamic analysis.

Similarly, self-adaptive systems may benefit from techniques developed for evolutionary trade-off analysis. These may include static or dynamic analysis of the system that can provide additional information to the bound analysis done by Thallium [4]. Any new information generated by such techniques will also help to reduce the run-time burden on the system.

Lastly, our presentation of Thallium demonstrated using a Pareto analysis to trim dominated adaptations from the search space. Other techniques also exist to perform such a multi-objective optimization, including game theoretic [56, 59] or stochastic [58] approaches. Depending on the level of trimming desired from the system, different heuristic could be applied to further reduce the number of adaptations considered at runtime. We would like to further explore these types of analysis which we believe would complement Thallium's use of the bound information.

We have made Thallium, as well as the specifications used in conducting our experiments, publicly available for use by other researchers [60].

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments. This work was supported in part by awards CCF-1755890 and CCF-1618132 from the National Science Foundation.

REFERENCES

[1] Jonathan Aldrich, David Garlan, Christian Kästner, Claire Le Goues, Anahita Mohseni-Kabir, Ivan Ruchkin, Selva Samuel, Bradley R. Schmerl, Christopher Steven Timperley, Manuela Veloso, Ian Voysey, Joydeep Biswas, Arjun Guha, Jarrett Holtz, Javier Cámara, and Pooyan Jamshidi. 2019. Model-Based Adaptation for Robotics Software. *IEEE Software* 36, 2 (2019), 83–90. https://doi.org/10.1109/MS.2018.2885058

[2] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49, 5 (2002), 672–713. https://doi.org/10.1145/585265.585270

[3] Martin Arlitt and Carey Williamson. 2004. Clark-Net HTTP. (2004). http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

[4] Hamid Bagheri and Sam Malek. 2016. Titanium: efficient analysis of evolving alloy specifications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 27–38. https://doi.org/10.1145/2950290.2950337

[5] Hamid Bagheri, Yuanyuan Song, and Kevin J. Sullivan. 2010. Architectural style as an independent variable. In *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, September 20-24, 2010*, Charles Pecheur, Jamie Andrews, and Elisabetta Di Nitto (Eds.). ACM, 159–162. https://doi.org/10.1145/1858996.1859026

[6] Hamid Bagheri and Kevin J. Sullivan. 2010. Monarch: Model-Based Development of Software Architectures. In *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part II (Lecture Notes in Computer Science)*, Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen (Eds.), Vol. 6395. Springer, 376–390. https://doi.org/10.1007/978-3-642-16129-2_27

[7] Hamid Bagheri and Kevin J. Sullivan. 2016. Model-driven synthesis of formally precise, stylized software architectures. *Formal Asp. Comput.* 28, 3 (2016), 441–467. https://doi.org/10.1007/s00165-016-0360-8

[8] Hamid Bagheri, Kevin J. Sullivan, and Sang H. Son. 2012. Spacemaker: Practical Formal Synthesis of Tradeoff Spaces for Object-Relational Mapping. In *Proceedings of the 24th International Conference on Software Engineering & Knowledge Engineering (SEKE'2012), Hotel Sofitel, Redwood City, San Francisco Bay, USA July 1-3, 2012*. Knowledge Systems Institute Graduate School, 688–693.

[9] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2014. TradeMaker: automated dynamic analysis of synthesized tradespaces. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 106–116. https://doi.org/10.1145/2568225.2568291

[10] Hamid Bagheri, Chong Tang, and Kevin J. Sullivan. 2017. Automated Synthesis and Dynamic Analysis of Tradeoff Spaces for Object-Relational Mapping. *IEEE Trans. Software Eng.* 43, 2 (2017), 145–163. https://doi.org/10.1109/TSE.2016.2587646

[11] Richard Bellman. 2010. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

[12] Craig Boutilier, Ronen I. Brafman, and Christopher Geib. 1998. Structured Reachability Analysis for Markov Decision Processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 24–32. http://dl.acm.org/citation.cfm?id=2074094.2074098

[13] Radu Calinescu, Simos Gerasimou, and Alec Banks. 2015. Self-adaptive Software with Decentralised Control Loops. In *Fundamental Approaches to Software Engineering - 18th International Conference, FASE 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings (Lecture Notes in Computer Science)*, Alexander Egyed and Ina Schaefer (Eds.), Vol. 9033. Springer, 235–251. https://doi.org/10.1007/978-3-662-46675-9_16

[14] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.* 37, 3 (2011), 387–409. https://doi.org/10.1109/TSE.2010.92

[15] Javier Cámara, Gabriel A. Moreno, and David Garlan. 2014. Stochastic Game Analysis and Latency Awareness for Proactive Self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, New York, NY, USA, 155–164. https://doi.org/10.1145/2593929.2593933

[16] Javier Cámara, Gabriel A. Moreno, David Garlan, and Bradley Schmerl. 2016. Analyzing Latency-Aware Self-Adaptation Using Stochastic and Simulations. *ACM Trans. Auton. Adapt. Syst.* 10, 4, Article 23 (Jan. 2016), 28 pages. https://doi.org/10.1145/2774222

[17] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. 2013. Automatic Verification of Competitive Stochastic Systems. *Formal Methods in System Design* 43, 1 (2013), 61–92.

[18] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Trans. Softw. Eng. Methodol.* 27, 2, Article 5 (June 2018), 50 pages. https://doi.org/10.1145/3204459

[19] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). 2009. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–26. https://doi.org/10.1007/978-3-642-02161-9_1

[20] Shang-Wen Cheng and David Garlan. 2012. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software* 85, 12 (2012), 2860–2875. https://doi.org/10.1016/j.jss.2012.02.060

[21] Shang-Wen Cheng, Vahe Poladian, David Garlan, and Bradley R. Schmerl. 2009. Improving Architecture-Based Self-Adaptation through Resource Prediction. In *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*. 71–88. https://doi.org/10.1007/978-3-642-02161-9_4

[22] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. 2018. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Asp. Comput.* 30, 1 (2018), 45–75. https://doi.org/10.1007/s00165-017-0432-4

[23] Zack Coker, David Garlan, and Claire Le Goues. 2015. SASS: Self-Adaptation Using Stochastic Search. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, Paola Inverardi and Bradley R. Schmerl (Eds.). IEEE Computer Society, 168–174. https://doi.org/10.1109/SEAMS.2015.16

[24] Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). 2013. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–32. https://doi.org/10.1007/978-3-642-35813-5_1

[25] Ahmed M. Elkhodary, Naeem Esfahani, and Sam Malek. 2010. FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, Gruia-Catalin Roman and André van der Hoek (Eds.). ACM, 7–16. https://doi.org/10.1145/1882291.1882296

[26] N. Esfahani, A. Elkhodary, and S. Malek. 2013. A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems. *IEEE Transactions on Software Engineering* 39, 11 (Nov 2013), 1467–1493. https://doi.org/10.1109/TSE.2013.37

[27] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming Uncertainty in Self-adaptive Software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 234–244. https://doi.org/10.1145/2025113.2025147

[28] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. 2011. Automated Verification Techniques for Probabilistic Systems. In *Formal Methods for Eternal Networked Software Systems (SFM'11) (LNCS)*, M. Bernardo and V. Issarny (Eds.), Vol. 6659. Springer, 53–113.

[29] Erik M. Fredericks. 2016. Automatically Hardening a Self-adaptive System Against Uncertainty. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '16)*. ACM, New York, NY, USA, 16–27. https://doi.org/10.1145/2897053.2897059

[30] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer* 37, 10 (Oct. 2004), 46–54. https://doi.org/10.1109/MC.2004.175

[31] Robert Givan, Thomas Dean, and Matthew Greig. 2003. Equivalence Notions and Model Minimization in Markov Decision Processes. *Artif. Intell.* 147, 1-2 (July 2003), 163–223. https://doi.org/10.1016/S0004-3702(02)00376-4

[32] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. 2008. A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In *Towards a Service-Based Internet, First European Conference, ServiceWave 2008, Madrid, Spain, December 10-13, 2008. Proceedings*. 122–133. https://doi.org/10.1007/978-3-540-89897-9_11

[33] Scott A. Hissam, Sagar Chaki, and Gabriel A. Moreno. 2015. High Assurance for Distributed Cyber Physical Systems. In *Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW '15)*. ACM, New York, NY, USA, Article 6, 4 pages. https://doi.org/10.1145/2797433.2797439

[34] Daniel Jackson. 2002. Alloy: A Lightweight Object Modelling Notation. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (April 2002), 256–290. https://doi.org/10.1145/505145.505149

[35] Jung Soo Kim and David Garlan. 2010. Analyzing architectural styles. *Journal of Systems and Software* 83, 7 (2010), 1216–1235. https://doi.org/10.1016/j.jss.2010.01.049

[36] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2015. A Survey on Engineering Approaches for Self-adaptive Systems. *Pervasive Mob. Comput.* 17, PB (Feb. 2015), 184–206. https://doi.org/10.1016/j.pmcj.2014.09.009

[37] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS)*, G. Gopalakrishnan and S. Qadeer (Eds.), Vol. 6806. Springer, 585–591.

[38] Young-Jou Lai and Ching-Lai Hwang. 1992. A New Approach to Some Possibilistic Linear Programming Problems. *Fuzzy Sets Syst.* 49, 2 (July 1992), 121–133. https://doi.org/10.1016/0165-0114(92)90318-X

[39] Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). 2019. *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. ACM. https://dl.acm.org/citation.cfm?id=3341527

[40] Ming Mao and Marty Humphrey. 2012. A Performance Study on the VM Startup Time in the Cloud. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing (CLOUD '12)*. IEEE Computer Society, Washington, DC, USA, 423–430. https://doi.org/10.1109/CLOUD.2012.103

[41] Andreas Metzger and Philipp Bohn. 2017. Risk-Based Proactive Process Adaptation. In *Service-Oriented Computing - 15th International Conference, ICSOC 2017, Malaga, Spain, November 13-16, 2017, Proceedings*. 351–366. https://doi.org/10.1007/978-3-319-69035-3_25

[42] Andreas Metzger, Osama Sammodi, and Klaus Pohl. 2013. Accurate Proactive Adaptation of Service-Oriented Systems. In *Assurances for Self-Adaptive Systems - Principles, Models, and Techniques*. 240–265. https://doi.org/10.1007/978-3-642-36249-1_9

[43] Gabriel A. Moreno. 2017. *Adaptation Timing in Self-Adaptive Systems*. Ph.D. Dissertation. School of Comp. Sci., Carnegie Mellon Univ., Pittsburgh, PA.

[44] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive Self-adaptation Under Uncertainty: A Probabilistic Model Checking Approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/2786805.2786853

[45] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2016. Efficient Decision-Making under Uncertainty for Proactive Self-Adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. 147–156. https://doi.org/10.1109/ICAC.2016.59

[46] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2018. Flexible and Efficient Decision-Making for Proactive Latency-Aware Self-Adaptation. *ACM Trans. Auton. Adapt. Syst.* 13, 1, Article 3 (April 2018), 36 pages. https://doi.org/10.1145/3149180

[47] Gabriel A. Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. DARTSim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems, See [39], 181–187. https://dl.acm.org/citation.cfm?id=3341554

[48] Gabriel A. Moreno, Bradley Schmerl, and David Garlan. 2018. SWIM: An Exemplar for Evaluation and Comparison of Self-adaptation Approaches for Web Applications. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '18)*. ACM, New York, NY, USA, 137–143. https://doi.org/10.1145/3194133.3194163

[49] Gabriel A. Moreno, Ofer Strichman, Sagar Chaki, and Radislav Vaisman. 2017. Decision-making with Cross-entropy for Self-adaptation. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '17)*. IEEE Press, Piscataway, NJ, USA, 90–101. https://doi.org/10.1109/SEAMS.2017.7

[50] Gustavo G. Pascual, Roberto E. Lopez-Herrejon, Mónica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying Multiobjective Evolutionary Algorithms to Dynamic Software Product Lines for Reconfiguring Mobile Applications. *J. Syst. Softw.* 103, C (May 2015), 392–411. https://doi.org/10.1016/j.jss.2014.12.041

[51] V. Poladian, D. Garlan, M. Shaw, M. Satyanarayanan, B. Schmerl, and J. Sousa. 2007. Leveraging Resource Prediction for Anticipatory Dynamic Configuration. In *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*. 214–223. https://doi.org/10.1109/SASO.2007.35

[52] prism doc 2019. PRISM Documentation. https://www.prismmodelchecker.org/doc/. (2019).

[53] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[54] Federico Quin, Danny Weyns, Thomas Bamelis, Sarpreet Singh Buttar, and Sam Michiels. 2019. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning, See [39], 1–12. https://dl.acm.org/citation.cfm?id=3341529

[55] Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. 2011. Plato: a genetic algorithm approach to run-time reconfiguration inÂăautonomic computing systems. *Cluster Computing* 14, 3 (01 Sep 2011), 229–244. https://doi.org/10.1007/s10586-010-0122-y

[56] Rui Meng, Ye Ye, and Neng-gang Xie. 2010. Multi-objective optimization design methods based on game theory. In *2010 8th World Congress on Intelligent Control and Automation*. 2220–2227. https://doi.org/10.1109/WCICA.2010.5554307

[57] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2, Article 14 (May 2009), 42 pages. https://doi.org/10.1145/1516533.1516538

[58] S. B. Selcuklu, D. W. Coit, F. Felder, M. Rodgers, and N. Wattanapongsakorn. 2013. A new methodology for solving multi-objective stochastic optimization problems with independent objective functions. In *2013 IEEE International Conference on Industrial Engineering and Engineering Management*. 101–105. https://doi.org/10.1109/IEEM.2013.6962383

[59] Kwee-Bo Sim, Ji-Yoon Kim, and Dong-Wook Lee. 2004. Game Theory Based Co-evolutionary Algorithm (GCEA) for Solving Multiobjective Optimization Problems. *IEICE Transactions* 87-D, 10 (2004), 2419–2425. http://search.ieice.org/bin/summary.php?id=e87-d_10_2419

[60] Clay Stevens and Hamid Bagheri. 2019. Thallium Website. https://sites.google.com/view/thallium/. (2019).

[61] Jacob Swanson, Myra B. Cohen, Matthew B. Dwyer, Brady J. Garvin, and Justin Firestone. 2014. Beyond the Rainbow: Self-adaptive Failure Avoidance in Configurable Systems. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 377–388. https://doi.org/10.1145/2635868.2635915

[62] Chong Tang, Hamid Bagheri, Sarun Paisarnsrisomsuk, and Kevin J. Sullivan. 2017. Towards designing effective data persistence through tradeoff space analysis. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard (Eds.). IEEE Computer Society, 353–355. https://doi.org/10.1109/ICSE-C.2017.106

[63] András Varga and Rudolf Hornig. 2008. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, Article 60, 10 pages. http://dl.acm.org/citation.cfm?id=1416222.1416290

[64] Chen Wang and Jean-Louis Pazat. 2012. A Two-Phase Online Prediction Approach for Accurate and Timely Adaptation Decision. In *2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, June 24-29, 2012*. 218–225. https://doi.org/10.1109/SCC.2012.26

[65] Danny Weyns. 2018. Engineering Self-Adaptive Software Systems - An Organized Tour. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, September 3-7, 2018*. 1–2. https://doi.org/10.1109/FAS-W.2018.00012

[66] Danny Weyns, Sam Malek, Rogério de Lemos, and Jesper Andersson (Eds.). 2010. *Self-Organizing Architectures, First International Workshop, SOAR 2009, Cambridge, UK, September 14, 2009, Revised Selected and Invited Papers*. Lecture Notes in Computer Science, Vol. 6090. Springer. https://doi.org/10.1007/978-3-642-14412-7

[67] L. A. Zadeh. 1999. Fuzzy Sets As a Basis for a Theory of Possibility. *Fuzzy Sets Syst.* 100 (April 1999), 9–34. http://dl.acm.org/citation.cfm?id=310817.310820

[68] H.-J. Zimmermann. 1996. *Fuzzy Set Theory&Mdash;and Its Applications (3rd Ed.)*. Kluwer Academic Publishers, Norwell, MA, USA.