Runtime Stability-Aware Adaptation for Self-Adaptive Systems using Symbiotic Simulation: A Cloud Architecture Evaluation

Maria Salama
School of Computer Science
University of Birmingham
Birmingham, UK
0000-0002-3184-683X

Rami Bahsoon
School of Computer Science
University of Birmingham
Birmingham, UK
0000-0002-1139-5795

Rajkumar Buyya
School of Computing and Information Systems
University of Melbourne
Melbourne, Australia
0000-0001-9754-6496

***Abstract*—Self-adaptive cloud architectures use runtime adaptation to maintain service quality under changing workloads, yet tactics restoring a violated objective may increase long-term instability through repeated reconfiguration or excessive overhead. Existing approaches assess adaptation success through immediate objective satisfaction, with limited support for runtime decision-making under uncertainty about whether a tactic will preserve stable behaviour over time. This paper presents a runtime stability-aware adaptation approach for self-adaptive systems, evaluated in the cloud architecture domain. Within an autonomic control loop, candidate tactics are evaluated through symbiotic *what-if* simulation before enactment, and the controller selects the tactic most likely to preserve service stability and adaptation quality. Implemented as an extension of *CloudSim* and evaluated with an auction-style benchmark under a bursty, real-world workload, the proposed controller reduces operational cost and peak resource utilisation by approximately 94% and adaptation frequency by 28%, whilst maintaining response time within the stability objective and achieving equivalent energy consumption. Interval-level analysis shows a reduction in response-time variability across all service types, indicating a stabilisation effect beyond mean response-time improvement. These results show that runtime stability-aware decision making can improve the efficiency and predictability of self-adaptation outcomes in a controlled cloud setting.**

***Index Terms*—architectural stability, runtime stability, self-adaptive software, cloud architecture, symbiotic simulation.**

I. Introduction

Self-adaptive cloud architectures operate under fluctuating workloads, changing service demands, and evolving runtime constraints [1]–[5]. To preserve service quality, they continuously monitor execution and trigger runtime adaptation actions [6], [7]. A central challenge is that adaptations effective in the short term may still degrade long-term behaviour. Frequent adaptation can itself become a source of instability through oscillation, excessive reconfiguration, or unnecessary resource allocation.

Villegas et al. [6] identify adaptation frequency, resource overshoot, settling time and accuracy as evaluation properties of self-adaptive systems, but use them as post-hoc measures rather than criteria governing tactic selection. Despite extensive work on self-adaptation and cloud resource management, runtime stability is rarely treated as an explicit decision criterion in tactic selection [1]–[3], [6]. Existing approaches typically assess adaptation success through immediate service quality recovery, utility optimisation, or compliance with service-level objectives (SLOs), without reasoning explicitly about whether the selected tactic will preserve stable behaviour over time or reduce future adaptation frequency. As a result, adaptation decisions may be locally effective whilst architecturally destabilising over time.

This paper addresses that gap through a runtime stability-aware adaptation approach for self-adaptive systems, evaluated in the domain of cloud architectures using symbiotic simulation. The novelty lies not in simulation for adaptation itself, but in using runtime stability explicitly as a tactic-selection criterion before enactment. The controller combines runtime monitoring with *what-if* simulation to evaluate candidate tactics prospectively, estimating their likely effect on both service stability and adaptation quality before any change is applied to the live system. It then selects the tactic most likely to preserve runtime stability over time. The approach is implemented as an extension of *CloudSim* [4], [8], a widely used discrete-event simulation toolkit for cloud computing environments, and evaluated using the *RUBiS* [9] benchmark, a multi-tier online auction application commonly used to generate representative e-commerce workloads, under dynamic workloads derived from the *World Cup 1998* trace [10], a widely used web-traffic trace characterised by a sharp surge in demand, comparing the proposed stability-aware controller against a baseline reactive self-adaptive controller.

The contributions of this paper are: (i) runtime stability model for self-adaptive cloud architectures that distinguishes service stability and adaptation quality; (ii) symbiotic simulation-based decision mechanism for stability-aware tactic selection, formalised as a five-phase runtime control loop; (iii) prototype implementation extending *CloudSim* with self-adaptation and stability-aware reasoning components; and (iv)

This work was partially funded by the School of Computer Science, University of Birmingham, U21 PhD scholarship, and the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, the University of Melbourne.

empirical evaluation using the *RUBiS* benchmark and the *World Cup 1998* workload trace, reporting results on service stability and adaptation quality, for two compared controllers.

The evaluation shows that the stability-aware controller reduces operational cost by approximately 94%, peak resource utilisation by 94%, and adaptation frequency by 28% relative to the baseline, whilst maintaining response time within the stability objective across all service configurations. Interval-level analysis further shows a reduction in response-time variability of 19–31% across all five service type configurations, indicating a stabilisation effect beyond mean response-time improvement. These results indicate that runtime stability-aware adaptation improves both service-level and adaptation-quality outcomes. The evaluation is intentionally scoped to a controlled simulation setting, providing an initial but bounded assessment of the proposed approach.

Organisation. Section II reviews background and related work. Section III defines runtime stability. Section IV presents the proposed approach. Section V describes the *CloudSim*-based implementation. Section VI reports the evaluation design and results. Section VII discusses implications and limitations. Section VIII concludes the paper.

II. Background and Related Work

A. Target Architecture Context

A self-adaptive cloud architecture comprises a managed system and an adaptation controller [6]. The managed system consists of physical hosts and virtual machines (VMs) that are deployed and reconfigured dynamically in response to runtime demand. The controller follows a variation of the MAPE-K feedback loop, monitoring the system, analysing its state against SLOs, planning adaptation actions, and executing selected tactics [6]. Typical tactics include vertical scaling, horizontal scaling, de-scaling and VM consolidation.

Cloud environments are inherently dynamic [5]. Workload fluctuation, changing quality requirements and peak demand make elasticity an operational requirement rather than an optional capability [5], [6]. Adaptation therefore affects both service behaviour and architectural configuration. Stability is consequently a dual concern: it is not enough to restore quality objectives momentarily; the architecture must do so without oscillation, excessive resource use, or repeated corrective actions that leave it persistently unsettled [6].

B. Runtime Evaluation in Self-Adaptive Systems

Self-adaptation has been widely motivated as a response to dynamic and uncertain operating environments [1]–[3]. These foundational works establish the MAPE-K feedback loop as the principal control structure for maintaining service quality objectives under changing workloads and environmental conditions, with some incorporating predictive or proactive elements into monitoring and planning. However, although candidate tactics may be selected using utility models, prediction, or optimisation, their effect on future runtime behaviour is rarely evaluated explicitly in terms of preserving runtime stability. Adaptation is typically judged by whether SLOs are satisfied, rather than by whether the chosen tactic is likely to maintain stable behaviour over time.

Villegas et al. [6] identify four evaluation properties of self-adaptive systems: adaptation frequency, resource overshoot, settling time and adaptation accuracy. The first two inform the adaptation-quality dimension in this paper directly; settling time and adaptation accuracy are discussed further in Section VII as candidate extensions. The fundamental distinction, however, is temporal: in [6] these properties are used as *post-hoc* evaluation measures after adaptation decisions have been enacted, whereas this paper uses them to inform tactic selection before an adaptation is committed.

Related work also addresses runtime evaluation through resilience, dependability and robustness. Ghosh et al. [11] evaluate cloud resilience using job rejection rate and response delay; Pataricza et al. [12] quantify resilience under environmental change; Gorbenko et al. [13], [14] study behavioural instability in service-oriented architectures; Almeida et al. [15] broaden resilience evaluation through changeload-based benchmarking; and Kounev et al. [16] survey dependable and resilient cloud provisioning. Although these approaches address properties related to stability, none integrates stability reasoning into the tactic-selection step of the adaptation controller.

C. Simulation for Adaptive Cloud Architectures

Symbiotic simulation [17]–[19] provides the foundation for the runtime reasoning mechanism proposed in this paper. In this approach, a simulation model runs in close coupling with a physical system, consumes real-time measurements, and returns feedback to guide operation. Its key capability is *what-if* reasoning: evaluating candidate decisions before they are applied to the live system. This is directly relevant to self-adaptive runtime control, where poor adaptation decisions may lead to SLO violations, instability, or unnecessary adaptation.

For self-adaptive systems, Abuseta et al. [20] proposed a simulation environment for testing systems designed around the IBM autonomic computing blueprint. Elhabbash et al. [21] further note the lack of simulation tools tailored to the design and evaluation of self-adaptive systems, with many existing environments focusing on hardware-centric or fixed-domain scenarios rather than dynamic, workload-driven settings.

D. Existing Approaches to Tactic Selection

A separate body of work addresses delayed tactic effects through probabilistic model checking rather than simulation, evaluating candidate adaptations over a look-ahead horizon using formal verification of a system model [22]–[24]. These approaches share this paper's concern with anticipating the future consequences of a tactic before it is applied, but rely on model verification rather than on simulating runtime dynamics.

In adaptive cloud management, several approaches address autonomous resource management and QoS optimisation. Brandic et al. [25] focus on compliance with SLOs; Gillam et al. [26] and Maurer et al. [27] address adaptive provisioning under dynamic workloads; and Jamshidi et al. [28] survey

cloud migration patterns and adaptation strategies. These approaches treat adaptation primarily as a QoS recovery mechanism, where the controller reacts to observed or predicted violations and selects tactics to restore compliance.

Positioning of This Paper. The literature reveals three gaps addressed by this paper. First, runtime evaluation in self-adaptive systems is well developed, but stability-related properties are assessed post-hoc, with the effect of a candidate tactic on future runtime behaviour rarely used as an explicit criterion for tactic selection. Second, although symbiotic simulation has proved useful for runtime decision support, it has not been explicitly used as a stability-aware adaptation mechanism within the MAPE-K control loop. Third, adaptive cloud management approaches usually optimise QoS provision without reasoning explicitly about the stability of the adaptation process itself. This paper addresses these gaps by introducing a runtime stability-aware adaptation mechanism for self-adaptive systems and evaluating it in the domain of cloud architectures. In this mechanism, runtime stability is treated explicitly as a tactic-selection criterion and reasoned about prospectively through symbiotic *what-if* simulation.

III. Runtime Stability of Self-Adaptive Cloud Architectures

Building on the dynamic operating conditions described above, runtime adaptation is essential for maintaining QoS [1]–[3], yet, as noted, restoring a violated quality attribute in the short term may still increase longer-term instability [6]. This section defines runtime stability formally and motivates its treatment as an explicit adaptation criterion.

A. Defining Runtime Stability

This paper is concerned with the operational perspective of stability — the architecture's ability to sustain intended behaviour during runtime under dynamic workload and continuous adaptation — rather than the evolutionary perspective (structural stability across software versions) [29].

Following the operational stability perspective in [29], runtime stability is defined as the ability of a self-adaptive cloud architecture to maintain critical service quality attributes within acceptable service-level bounds over time whilst avoiding excessive, oscillatory, or ineffective adaptation actions. Runtime stability is therefore not the absence of change. A cloud architecture may need to adapt repeatedly to remain effective; stability instead concerns responding to change in a controlled and convergent way without allowing adaptation itself to become a source of degradation.

Runtime stability has two complementary dimensions: (i) service stability, concerning whether service quality attributes remain within their SLOs over time; and (ii) adaptation quality, concerning whether the adaptation process converges efficiently without repeated, ineffective, or unnecessarily frequent reconfiguration actions. An architecture may appear stable from a service perspective if it eventually restores response time after each disturbance, yet still exhibit poor adaptation quality if doing so requires frequent reconfiguration or long settling periods. Conversely, a controller may minimise adaptation frequency but fail to preserve service quality under sustained demand changes. Runtime stability therefore requires reasoning about both dimensions together.

Service stability refers to the sustained provision of service quality attributes relevant to stakeholders, including performance attributes (e.g., response time and throughput) and operational concerns (e.g., energy consumption and cost). A service is stable when these attributes remain within their SLOs over time. Service stability therefore concerns maintaining multiple service quality attributes within acceptable bounds whilst recognising possible trade-offs during peak demand and recovery periods. As with the adaptation-quality properties (discussed in Section II-B), this paper treats service quality prospectively, as a criterion used during reasoning before adaptation is committed, rather than as a post-hoc evaluation measure.

Adaptation quality refers to the quality of the adaptation process itself. Since self-adaptive cloud architectures continuously monitor and reconfigure their deployment, adaptation becomes an architectural concern in its own right. Adaptation is of high quality when it converges the architecture efficiently towards its objectives without repeated or unnecessary reconfiguration. A high-quality adaptation process should converge within a reasonable time, avoid repeated adaptation cycles indicating short-lived or ineffective tactics, minimise unnecessary resource overshoot, and incur only modest reasoning overhead.

B. Stability Objectives for Runtime Adaptation

Based on the above definition, runtime stability in the cloud setting is assessed with respect to two broad classes of objectives.

Service-oriented objectives concern the sustained satisfaction of critical runtime quality objectives, including QoS targets, SLOs and service quality attributes (e.g., performance, availability, responsiveness). Such objectives are expressed as acceptable operating bounds over time.

Adaptation-oriented objectives concern the quality of the adaptation process itself, including timely recovery, convergence, avoidance of repeated or unnecessary adaptation cycles, and limitation of resource overshoot and computational overhead.

These objectives may interact and sometimes conflict. Increasing resources early may improve response time and reduce settling time, but may also increase energy consumption and temporary overshoot. Runtime adaptation must therefore reason about trade-offs rather than optimise a single metric in isolation. In self-adaptive systems, carefully selected reconfiguration may be necessary to preserve stable service behaviour. The challenge is not to avoid adaptation, but to select adaptations that improve runtime conditions without causing repeated future instability. The specific quality attributes, thresholds and evaluation measures that instantiate these objectives depend on the application context.

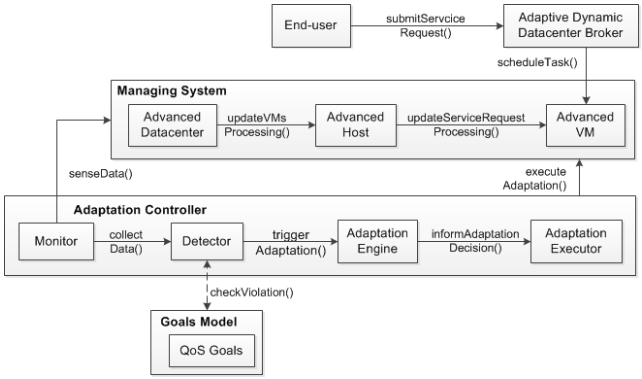


Fig. 1: Runtime stability-aware adaptation process

C. The Need for Explicit Runtime Stability Reasoning

A key limitation of many self-adaptive approaches is that they treat adaptation success primarily in terms of immediate objective satisfaction [1]–[3]. If a violation is detected, the controller selects a tactic intended to restore the violated objective, often with limited reasoning about how that tactic may affect future runtime behaviour. This may lead to locally effective but globally unstable behaviour [6], [29]. An aggressive scaling action may remove an immediate bottleneck but later require repeated compensatory actions, whereas a minimal reaction may leave the system in a prolonged degraded state.

Making runtime stability explicit shifts the decision problem from *which tactic fixes the current violation* to *which tactic is most likely to restore acceptable operation whilst preserving stable behaviour over time*. This is particularly relevant in cloud architectures, where delayed tactic effects [22]–[24], workload bursts and resource interdependencies make the consequences of runtime decisions difficult to predict from current observations alone. Addressing this limitation requires a means of evaluating the likely future consequences of candidate tactics before enactment.

Problem Statement. The problem addressed in this paper is therefore: given a self-adaptive cloud architecture operating under dynamic workload conditions, how can the adaptation controller select runtime tactics that not only restore SLOs, but also preserve behavioural stability over time? To answer this question, the remainder of the paper presents a runtime stability-aware adaptation approach in which candidate tactics are evaluated through symbiotic what-if simulation before enactment. The goal is to support decisions that improve both service stability and adaptation quality, rather than reacting only to immediate runtime violations.

IV. Runtime Stability-Aware Adaptation

This section presents the proposed runtime stability-aware adaptation mechanism. Symbiotic simulation provides the runtime decision-support mechanism for this pre-enactment evaluation. Figure 1 illustrates the system components and adaptation process.

A. Controller Architecture

The approach is structured as a closed-loop controller extending a foundational self-adaptation controller [1], [2] and comprises the following components:

- *Monitor*: observes the live system state and maintains adaptation history.
- *Detector*: evaluates monitored values against stability objectives and raises an adaptation trigger on current or predicted violation.
- *Symbiotic Simulation*: seeds the simulation model from the current runtime state $S(t)$ and evaluates each candidate tactic before enactment, returning predicted post-tactic states.
- *Adaptation Engine*: filters and ranks candidate tactics using a service filter followed by stability-aware ranking.
- *Adaptation Executor*: applies the selected tactic to the live infrastructure and records it in the adaptation history.

B. Stability-Aware Adaptation Process

Runtime monitoring. At each observation interval, the *Monitor* collects: (i) workload characteristics (e.g., request arrival rate and service composition); (ii) service quality attributes (e.g., response time); (iii) the current architectural configuration (e.g., number and type of active hosts and VMs); and (iv) adaptation history $A(t)$. The adaptation history supports stability-aware reasoning by indicating whether a candidate tactic is likely to result in repeated adaptations.

Triggering adaptation. Adaptation is triggered when the *Detector* identifies either a current service-objective violation or a predicted near-term violation. A reactive trigger is raised when a monitored service quality indicator exceeds its configured objective θ, and a proactive trigger is raised when workload projection indicates that a violation is likely in the next interval. This workload projection informs triggering only; it is distinct from the Symbiotic Simulation's tactic evaluation described below, which assesses each candidate tactic's predicted effect under the currently observed workload rather than a projected one. When no trigger fires, no adaptation is enacted.

Candidate tactics. The tactic space $\mathcal{T}$ comprises architectural reconfiguration actions for restoring service quality objectives. These typically target resource provisioning (e.g., scaling or consolidation). Each tactic $\tau \in \mathcal{T}$ may have multiple variants parameterised by magnitude of change, forming the candidate set passed to the *Symbiotic Simulation*.

Symbiotic what-if evaluation. Because the controller cannot directly observe the future effect of a tactic without enacting it [17]–[19], symbiotic simulation is used to reduce this uncertainty. At each adaptation trigger, the *Symbiotic Simulation* seeds the model from current runtime observations — including the currently observed workload $W(t)$ — and evaluates each candidate tactic, producing a predicted post-tactic state $\hat{S}(t+\Delta t \mid \tau_i)$. The workload is held constant at $W(t)$ throughout the look-ahead horizon: the simulation therefore isolates the predicted effect of the candidate tactic itself under sustained current demand, rather than forecasting how

TABLE I: Notation used in Algorithm 1

Symbol	Description
$S(t)$	Observed system state at time t: $\{Q(t), H(t), V(t)\}$
$A(t)$	Adaptation history up to time t
$W(t)$	Observed workload at time t
$\mathcal{T}$	Set of available candidate tactics
Q	Set of service quality attributes
Θ	Set of stability objectives $\{\theta_j \mid q_j \in Q\}$
τ^*	Selected optimal tactic
$\varnothing$	No adaptation
$H(t)$	Number of active hosts at time t
$V(t)$	Number and type of active VMs at time t
$q_j(t)$	Observed value of quality attribute $q_j \in Q$ at time t
θ_j	Stability objective for quality attribute q_j
$\hat{S}(t+\Delta t \mid \tau_i)$	Predicted post-tactic state for candidate tactic τ_i
E	Eligible set of candidate tactics satisfying all service objectives
w_j	Weight for service-quality attribute q_j
n	Number of adaptation-quality attributes considered
f_k	Adaptation-quality attribute k ($k = 1, \ldots, n$)

future workload change might interact with that tactic. These predicted states estimate the expected effect of each tactic on both service and adaptation quality under this assumption, before any change is applied to the live system.

Stability-aware tactic selection. The *Adaptation Engine* first filters candidate tactics $\tau_i \in \mathcal{T}$ whose predicted service quality still violates the objective θ. Only tactics that restore service objectives form the eligible set E; if $E = \emptyset$, the candidate with the lowest weighted service-quality violation, $\sum_j w_j \hat{q}_j$, is selected as a best-effort fallback. Each eligible candidate $\tau_i \in E$ is then scored against n adaptation-quality attributes $f_1, \ldots, f_n$, drawn from properties such as those identified by Villegas et al. [6] or other related concerns such as reasoning and enactment overhead. Each f_k is normalised over the candidate set at the current control step. The candidate with the lowest aggregate adaptation-quality score is selected:

$$\tau^* = \arg\min_{\tau_i \in E} \sum_{k=1}^{n} \hat{f}_k(\tau_i) \tag{1}$$

where $\hat{f}_k(\tau_i) \in [0, 1]$ is the min-max normalised value of attribute k for candidate τ_i, computed over the current candidate set at each control step. The n attributes are combined without differentiation, consistent with their treatment as a single aggregate allocation.

C. Runtime Control Loop

Algorithm 1 (Table I summarises the notation) formalises the stability-aware runtime control loop. A deployment may instead weight individual attributes or select a different attribute set depending on the properties of the target system.

V. CloudSim-based Symbiotic Simulation Environment

The symbiotic simulation capability is realised as a runtime decision-support layer coupled with the live environment. At each adaptation trigger, it instantiates a separate *CloudSim* instance, seeded from the current runtime state $S(t)$ and workload $W(t)$, in which each candidate tactic is evaluated before being returned to the *Adaptation Engine*. Figure 2 illustrates the concept of symbiotic simulation.

Algorithm 1 Stability-Aware Runtime Control Loop

Input: $S(t)$, $A(t)$, $W(t)$, $\mathcal{T}$, Q, Θ
while system is operating **do**
  // Phase 1 — Monitor
  $S(t) \leftarrow \{Q(t), H(t), V(t)\}$
  $W(t) \leftarrow$ current workload arrival rate and service composition
  $A(t) \leftarrow$ retrieve adaptation history up to time t
  // Phase 2 — Detect
  $violated \leftarrow$ **false**
  if $\exists\, q_j \in Q : \text{violates}(q_j(t), \theta_j)$ **then**
    $violated \leftarrow$ **true**
  else
    if $\text{predictViolation}(S(t), W(t)) =$ **true then**
      $violated \leftarrow$ **true**
    end if
  end if
  if $violated =$ **true then**
    // Phase 3 — Symbiotic What-If Evaluation
    $\tau^* \leftarrow \varnothing$
    $candidates \leftarrow \text{generateCandidates}(\mathcal{T}, S(t), W(t))$
    for $\tau_i \in candidates$ **do**
      $\hat{S}(t+\Delta t \mid \tau_i) \leftarrow \text{symbioticSimulate}(\tau_i, S(t), W(t))$
    end for
    // Phase 4 — Stability-Aware Ranking and Selection
    $E \leftarrow \{\tau_i \mid \forall q_j \in Q : \neg\text{violates}(\hat{q}_j(t + \Delta t \mid \tau_i), \theta_j)\}$
    if $E = \emptyset$ **then**
      $\tau^* \leftarrow \arg\min_{\tau_i}\ \sum_j w_j\, \hat{q}_j(t+\Delta t \mid \tau_i)$
    else
      for $\tau_i \in E$ **do**
        $score(\tau_i) \leftarrow \sum_{k=1}^{n} \hat{f}_k(\tau_i)$
      end for
      $\tau^* \leftarrow \arg\min_{\tau_i \in E}\ score(\tau_i)$
    end if
    // Phase 5 — Execute
    apply τ^* to live infrastructure
    append τ^* to $A(t)$
  end if
  wait Δt
end while

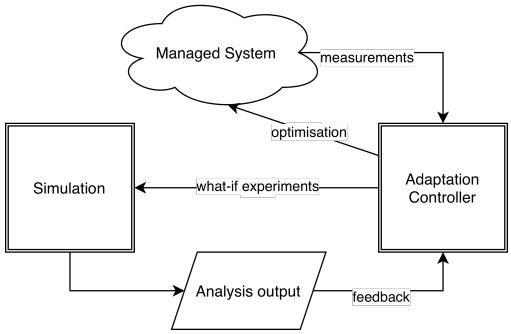


Fig. 2: Symbiotic simulation

The symbiotic simulator was built as an extension of the widely adopted *CloudSim* platform [4], [8]. *CloudSim* is a modular, discrete-event simulation environment for cloud computing research [30], [31], making it well suited to the addition of self-adaptation and stability-aware reasoning components. The proposed environment extends *CloudSim* with components for self-adaptation, dynamic workloads and runtime stability-aware reasoning. Figure 3 shows the architecture of the extended simulation environment.

The following core *CloudSim* components were extended: (i) *Datacenter*: QoS attribute tracking at the datacenter level; (ii) *Host*: per-host energy consumption using the power models of [32]; (iii) *VM*: per-VM resource utilisation and cost tracking based on Amazon EC2 pricing; and (iv) *DatacenterBroker*:

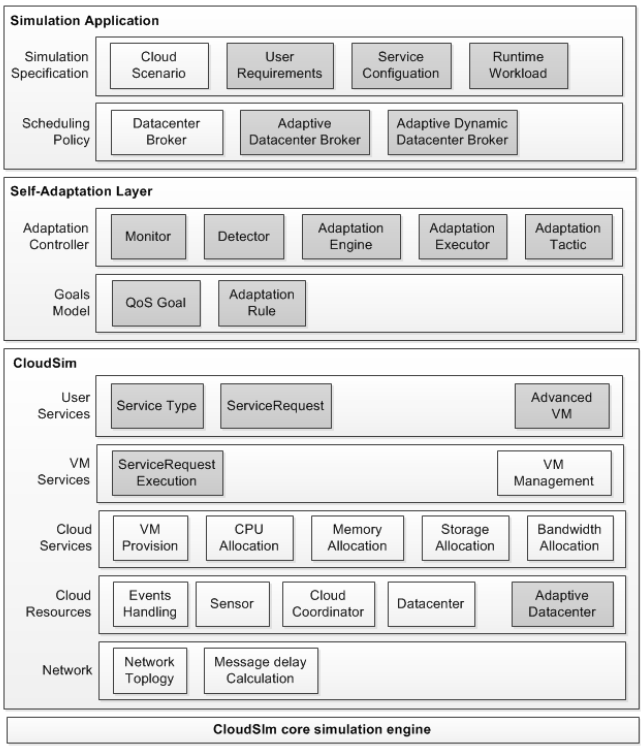


Fig. 3: Self-adaptive CloudSim architecture

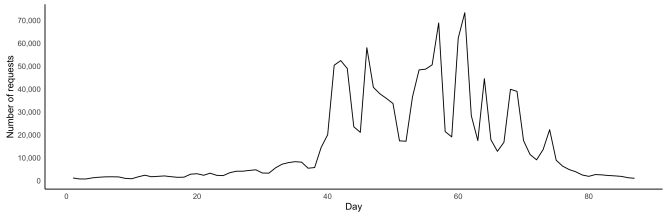


Fig. 4: World Cup 1998 workload trend.

TABLE II: Experimental setup

Parameter	**Value**	
Service Type Configurations		
S#1	Browsing only (read-only)	10,000 MIPS
S#2	Bidding only (read & write)	20,000 MIPS
S#3	Mixed 70% browsing, 30% bidding	12,000 MIPS
S#4	Mixed 50% browsing, 50% bidding	15,000 MIPS
S#5	Mixed 30% browsing, 70% bidding	17,000 MIPS
Infrastructure		
No. of hosts	10 running (max. 1,000)	
Host type	IBM x3550 server	
Host specification	2 × Xeon X5675, 3,067 MHz, 6 cores, 256 GB RAM	
CPU mapping	3,067 MIPS per core [32]	
Energy model	Beloglazov power models [32]	
No. of VMs	15 (5 × each type)	
VM types	General Purpose Amazon EC2 instances [33]: m4.large (2 vCPU, 8 GB, $0.10/hr) m4.xlarge (4 vCPU, 16 GB, $0.20/hr) m4.2xlarge (8 vCPU, 32 GB, $0.40/hr)	
Max. capacity	1,000 hosts	

queuing models for workload distribution and resource provisioning, providing the hooks required by the *Monitor* and *Adaptation Executor*. Two additional classes support dynamic workload modelling: (i) *Service* class models a SaaS service, parameterised by its computational requirement (MIPS); and (ii) *ServiceRequest* class models an end-user request for a specific service, enabling simulation of dynamic workloads across multiple users and service types. The stability-aware controller is implemented as an *Adaptation* package added to the *CloudSim* core.

VI. Empirical Evaluation

The empirical evaluation assesses whether runtime stability reasoning improves service-level stability and adaptation quality. The evaluation is conducted through simulation using *CloudSim* [8], which provides a controlled and repeatable environment for cloud experimentation. Two controller configurations are compared under realistic cloud workload conditions: a baseline reactive controller and the proposed stability-aware controller. The evaluation considers two dimensions: (i) sustained satisfaction of SLOs relative to the baseline controller; and (ii) quality of adaptation, reflected in adaptation frequency, overhead and resource overshoot.

A. Experimental Setup

To stress the architecture under realistic, variable demand, the evaluation uses the *RUBiS* benchmark [9] driven by the *World Cup 1998* workload trace [10].

Workload. RUBiS is an online auction application whose services fall into two workload patterns: *browsing*, comprising read-only services (e.g., `BrowseCategories`), and *bidding*, comprising read- and write-intensive services (e.g., `PutBid`, `RegisterItem` and `RegisterUser`). Each service is mapped to a Million Instructions Per Second (MIPS) rating. Five service type configurations are evaluated, ranging from browsing-only to bidding-only and three mixed compositions, as listed in Table II. Request volume was varied proportionally according to the *World Cup 1998* trend [10], which exhibits sustained low demand in the early phase followed by sharp volatility from day 40 onwards, with peak request volumes exceeding 70,000 per day. Figure 4 illustrates the workload trend, showing a prolonged low-demand phase followed by a highly volatile period with pronounced peaks in request volume.

Infrastructure configuration. The initial deployment and infrastructure configuration are shown in Table II.

Runtime Stability Objectives. The stability objectives instantiated in the experiments, listed in Table III, are specific to this cloud deployment scenario. The service-level attributes capture both SLOs and operational concerns: response time is the primary SLO; energy consumption captures environmental impact in kWh; and operational cost captures VM running cost per control interval. The weighting scheme prioritises SLO satisfaction, followed by environmental and economic concerns, with the remaining weight ($w_4 = 0.10$) assigned to the composite adaptation-quality score; w_4 is thus the residual

TABLE III: Stability attributes instantiated in the experiments

Attribute	Weight (w_i)	Metric	Objective
Service-level			
Response time	$w_1 = 0.50$	ms	≤ 25
Energy consumption	$w_2 = 0.20$	kWh	≤ 25
Operational cost	$w_3 = 0.20$	$	≤ 50
Adaptation quality			
Adaptation frequency		cycles	—
Adaptation overhead	$w_4 = 0.10$	seconds	—
Resource overshoot		hosts, VMs	—

TABLE IV: Architectural tactic space $\mathcal{T}$

Tactic	Description
Vertical scaling	Increase the number of VMs or their CPU capacities
Vertical de-scaling	Decrease the number of VMs or their CPU capacities
Horizontal scaling	Increase the number of running hosts
Horizontal de-scaling	Decrease the number of running hosts
VM consolidation	Migrate VMs to fewer hosts and shut down idle hosts
Concurrency	Increase thread-level parallelism in request processing
Dynamic scheduling	Apply priority-based scheduling policy

weight after service-level attributes have been prioritised, rather than an independently tuned parameter. The $n = 3$ adaptation-quality attributes are: adaptation frequency f_1, capturing whether the simulated trajectory is likely to trigger a further adaptation within the look-ahead horizon, informed by the recent adaptation history $A(t)$; adaptation overhead f_2, estimating the cost of monitoring, reasoning and enactment associated with the tactic; and resource overshoot f_3, measuring excess allocated hosts and VMs over the horizon. The same attributes were used across all configurations and were not tuned per workload or controller. The thresholds are intentionally challenging relative to the workload, to expose meaningful trade-offs between service-level and adaptation-quality objectives, whilst remaining representative of realistic cloud deployment constraints. The sensitivity of the results to the adaptation-quality attributes and to the look-ahead horizon length is not evaluated in this study; both are treated as fixed design choices, and their systematic variation is identified as a direction for future work.

Adaptation Tactics. The tactic space $\mathcal{T}$ comprises the architectural tactics listed in Table IV. Tactics addressing response time violations operate through vertical and horizontal scaling; energy consumption is targeted through VM consolidation; scheduling and concurrency provide additional response time management mechanisms. Adaptation rules associate tactics with stability attributes and define priority order, as listed in Table V. For the baseline controller, these rules determine tactic selection directly. For the stability-aware controller, they constrain the candidate set passed to the symbiotic simulation.

Compared Controllers. Both controllers use the same tactic space $\mathcal{T}$, workload, trigger logic and execution environment, and are evaluated under identical service objectives; they differ only in tactic selection. The baseline reactive controller enacts the first applicable tactic according to the priority order in Table V, without look-ahead evaluation. The reactive baseline was chosen to isolate the effect of stability-aware tactic selection under controlled conditions, rather than to represent the strongest available adaptation strategy. The stability-aware controller applies the same rules to identify candidates, then evaluates them through symbiotic what-if simulation over a three-interval horizon ($3 \times \Delta t = 2{,}592$ seconds) and enacts the candidate that minimises the stability-aware ranking function. The observed gains therefore arise from differences in tactic selection rather than from any relaxation of service objectives.

TABLE V: Adaptation rules

Tactic	Stability attribute	Priority
Dynamic scheduling	Response time	1
Concurrency	Response time	2
Vertical scaling	Response time	3
Horizontal scaling	Response time	4
VM consolidation	Energy consumption	1
Vertical de-scaling	Operational cost, energy	2
Horizontal de-scaling	Operational cost, energy	3

Simulation environment. The simulation was executed on an Intel Core i5 2.9 GHz machine with 16 GB RAM running Java JDK 1.8. Each controller was evaluated across all five service type configurations (S#1–S#5), yielding ten experimental conditions, with 30 independent runs executed per condition. In each run, the full *World Cup 1998* trace was executed at the confirmed compression ratio of $\Delta t = 864$ seconds per control interval. The symbiotic simulation component ran as a separate *CloudSim* instance concurrently with the main simulation. Results are reported as means across these runs, compared per configuration.

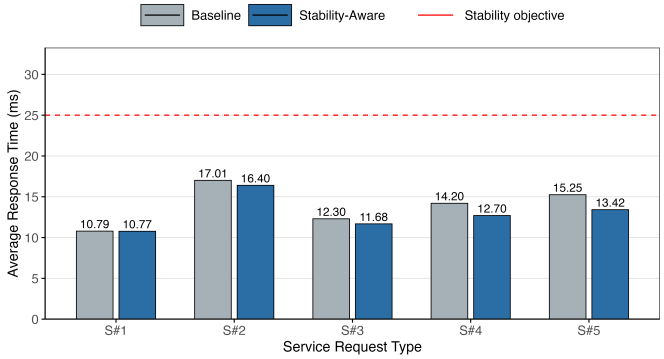
B. Service-level Results

Figure 5 shows the average response time and total operational cost for each service type configuration.

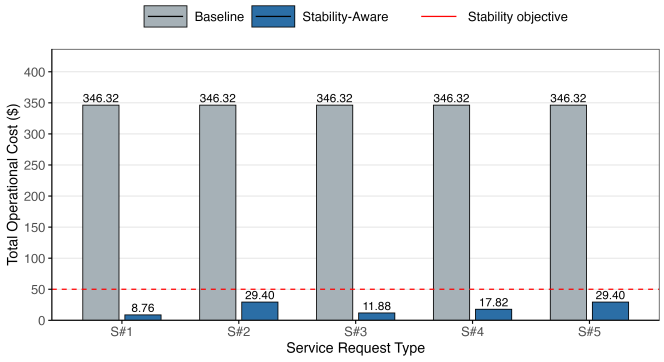
Response time. Both controllers satisfy the 25 ms stability objective across all five service type configurations. The stability-aware controller achieves lower average response time in all configurations, with overall means of 12.99 ms and 13.91 ms for the stability-aware and baseline controllers, respectively. The largest differences occur in write-intensive configurations: S#2 reduces from 17.01 ms to 16.40 ms, S#4 from 14.20 ms to 12.70 ms, and S#5 from 15.25 ms to 13.42 ms, whilst the difference is negligible in S#1 (10.79 ms vs. 10.77 ms).

Operational cost. The baseline controller incurs $346.32 across all five configurations, exceeding the $50 stability objective by a factor of approximately seven. The stability-aware controller remains within the $50 objective across all configurations, with an average cost of $19.45, a reduction of approximately 94%. Costs are lowest in S#1 ($8.76) and S#3 ($11.88), and highest in S#2 and S#5 ($29.40 each), consistent with the greater resource demand of write-intensive workloads.

Energy consumption. Aggregate energy consumption is 67.59–67.60 kWh for both controllers across all service type configurations, with no meaningful difference at the total level. This is consistent with the experimental configuration: energy is calculated at the host level using the Beloglazov power models [32], which are sensitive to host-level CPU utilisation



(a) Average response time.



(b) Total operational cost.

Fig. 5: Average (a) response time and (b) total operational cost per service type.

but insensitive to the number of VMs allocated per host. The substantial cost difference therefore reflects VM-level allocation decisions that the host-level energy model does not capture. The stability-aware controller therefore improves response time, cost and adaptation frequency without additional energy expenditure.

Overall, the stability-aware controller satisfies both the response-time and operational-cost objectives across all five configurations. The baseline satisfies the response-time objective but fails the cost objective in every configuration, with identical cost values ($346.32) regardless of service composition. This structural insensitivity to workload type indicates that the baseline's cost overrun arises from its tactic-selection logic rather than from workload demand, and that the stability-aware controller's reductions, ranging from 91.5% to 97.5% across configurations, are attributable to stability-aware ranking rather than to favourable workload conditions. The direction of improvement is consistent across all five configurations for both metrics.

To examine whether these aggregate service-level differences also hold during volatile workload periods, Figure 6 and Table VI report interval-level response-time behaviour.

Figure 6 shows mean response time with ± 1 standard deviation bands over the peak-demand phase (control intervals 35–70) for both controllers across all five service type configurations. This interval range corresponds to the period

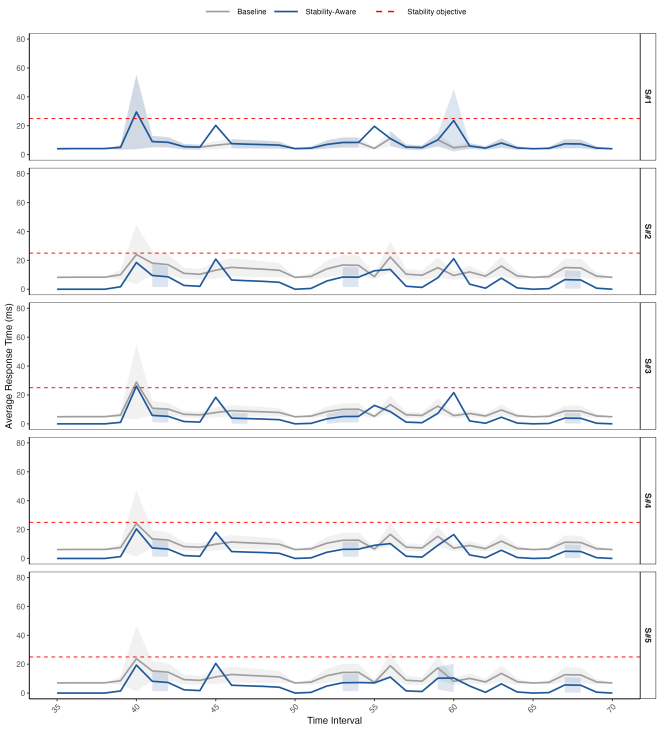


Fig. 6: Mean response time with ± 1 standard deviation over peak-demand intervals (35–70).

TABLE VI: Response-time statistics across all time intervals

Svc.	Mean RT (ms)			SD (ms)		
	Base.	Stab.	Δ%	Base.	Stab.	Δ%
S#1	6.09	6.09	−0.0	2.64	2.13	−19.3
S#2	10.98	2.51	−77.1	3.54	2.72	−23.2
S#3	7.15	2.05	−71.3	2.85	2.14	−24.9
S#4	8.63	2.05	−76.2	3.11	2.16	−30.5
S#5	9.54	2.08	−78.2	3.22	2.22	−31.1

of sharp workload variability in the *World Cup 1998* trace, where adaptation decisions have the greatest effect on service stability. The figure reveals two patterns not visible in the aggregate bar chart. First, the divergence between the two controllers is concentrated at workload spikes, most prominently around interval 40 and, to a lesser extent, intervals 55–60, where the stability-aware controller consistently suppresses peak response times more effectively. Second, the stability-aware controller produces narrower standard deviation bands throughout the volatile phase, indicating more predictable behaviour under sustained demand fluctuation.

Table VI complements Figure 5(a) by reporting mean response time and standard deviation computed from interval-level request data across all 87 control intervals. Whilst the bar chart reports per-configuration run means, the table captures the distribution of response times across individual control intervals, providing a finer-grained view of variability within each configuration.

The stability-aware controller reduces mean response time by 71–78% in write-intensive configurations (S#2–S#5) and

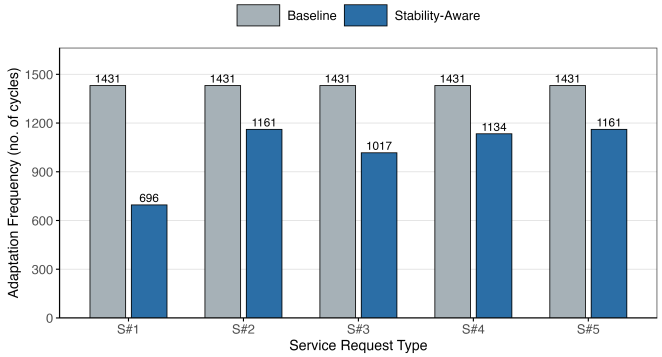


Fig. 7: Adaptation frequency per service type.

reduces response-time variability by 19–31% across all five configurations. The reduction in standard deviation is consistent even in S#1, where the mean response-time difference is negligible. In the read-only browsing configuration, both controllers select broadly similar tactics under low load, but the stability-aware controller's look-ahead tactic evaluation still produces more consistent interval-level behaviour. The standard deviation reduction across all configurations therefore indicates a genuine stabilisation effect beyond the mean response-time improvement.

C. Quality of Adaptation Results

Adaptation frequency. Figure 7 shows adaptation frequency per service type configuration, reported here as the realised number of adaptation cycles over the full evaluation — distinct from the predictive, per-candidate frequency estimate f_1 used in ranking (Section VI-A). The baseline controller executes 1,431 adaptation cycles across all five configurations, reflecting its reactive response to every workload fluctuation regardless of service characteristics. The stability-aware controller reduces adaptation frequency in all configurations, with an average of 1,034 cycles, a reduction of approximately 28%. The reduction is largest in S#1 (696 cycles) and smallest in S#2 and S#5 (1,161 cycles each), consistent with the higher MIPS demand of write-intensive configurations.

Adaptation overhead. Measured as the total time spent by the runtime control loop in monitoring, deciding and executing adaptation actions over the full evaluation — distinct from the predictive, per-candidate overhead estimate f_2 used in ranking (Section VI-A) — adaptation overhead is lower in aggregate for the stability-aware controller than for the baseline across all five service type configurations. The baseline incurs 206.30 seconds uniformly across all configurations, whereas the stability-aware controller averages 193.74 seconds, ranging from 188.60 seconds (S#1) to 197.30 seconds (S#2 and S#5). In this study, the lower aggregate adaptation time is consistent with the reduced adaptation frequency of the stability-aware controller. However, the isolated computational cost of the symbiotic *what-if* simulation phase was not measured independently, so no stronger claim can be made about the cost of the simulation phase.

TABLE VII: Summary of results across evaluation dimensions

Metric	Baseline	Stability-aware	Change
Service-level stability			
Avg. response time (ms)	13.91	12.99	−6.6%
Total operational cost ($)	346.32	19.45	−94.4%
Total energy (kWh)	67.60	67.60	≈ 0%
Adaptation quality			
Adaptation frequency (cycles)	1,431	1,034	−27.7%
Adaptation overhead (s)	206.30	193.74	−6.1%
Peak hosts (overshoot)	333	20	−94.0%
Peak VMs (overshoot)	333	19	−94.3%

Resource overshoot. Measured as the number of hosts and VMs utilised by the adaptation process, resource overshoot also differs substantially between the two controllers. The baseline controller reaches a peak of 333 hosts and 333 VMs across all five service type configurations. The stability-aware controller limits peak utilisation to 20 hosts and 19 VMs across all configurations, a reduction of approximately 94%. This reduction is consistent across all service type configurations and directly explains the operational-cost reduction reported in Section VI-B.

Across all five service type configurations and all reported metrics, the stability-aware controller produces lower values than the baseline without exception, providing consistent empirical support for the contribution under the evaluated conditions.

D. Overall Summary

Table VII summarises the results across the two evaluation dimensions. On the reported metrics, the stability-aware controller introduces no clear penalty relative to the baseline. In the evaluated setting, the additional reasoning introduced by *what-if* evaluation is offset at the aggregate level by the reduction in adaptation frequency, yielding lower total adaptation time. A more precise decomposition of per-cycle reasoning cost remains future work.

VII. Discussion

Interpreting the Results. The stability-aware controller's gains stem from evaluating candidate tactics before enactment and preferentially selecting those whose predicted effects persist across the look-ahead horizon, avoiding the repeated scaling and de-scaling cycles that accumulate resource expenditure under purely reactive control. Response time remains within the stability objective throughout and response-time variability falls by 19–31% across all five service type configurations — a stabilisation effect observed even in S#1, where mean response times are similar across both controllers. The contribution therefore lies in achieving acceptable service behaviour more sustainably and predictably, not in recovering from failures the baseline cannot handle. The equivalent aggregate energy result confirms this is not achieved at environmental expense. The cost and overshoot reductions arise from avoiding repeated overscaling rather than relaxing SLOs, since both controllers satisfy the same response-time objective under identical conditions, with the stability-aware controller doing so using substantially fewer resources. Runtime reconfiguration is not the

problem this approach addresses; *ineffective* reconfiguration is — and the prospective evaluation mechanism is what prevents it.

Runtime Stability Model. The adaptation-quality attributes instantiated in this work are adaptation frequency f_1, adaptation overhead f_2 and resource overshoot f_3; overhead reflects the cost of monitoring, reasoning and enactment introduced by the symbiotic simulation mechanism itself, rather than being drawn from Villegas et al. [6]. Of the remaining two, f_1 and f_3 correspond directly to two of the four stability properties Villegas et al. identify. The other two Villegas properties, settling time and adaptation accuracy, are not instantiated as scoring attributes in this study. Adaptation accuracy is addressed indirectly: candidate tactics violating the objective θ are excluded during eligibility checking, which distinguishes compliant from non-compliant tactics but does not rank compliant tactics by how closely they converge to the target. An explicit accuracy attribute, scoring the distance between a tactic's predicted outcome and a target operating point, could be added, as could settling time, estimating the number of simulated intervals required to return within the service objective after applying a tactic.

Computational Cost of Symbiotic Simulation. The proposed *what-if* evaluation mechanism trades the low computational cost of earlier offline analytical approaches [34] for the higher representational fidelity of *CloudSim*-based symbiotic simulation, a trade-off closer in kind to probabilistic model-checking approaches to proactive adaptation [22]–[24], which similarly reason over a look-ahead horizon and report verification time as part of their evaluation. The aggregate adaptation time in Table VII includes the simulation phase, but it was not timed independently, so this study cannot quantify the cost of a single what-if evaluation or how it scales with candidate-set size — currently fixed by the adaptation rules (Table V) rather than varied experimentally. Characterising this trade-off remains future work.

Applicability. Although evaluated in a cloud setting, the approach applies more broadly to self-adaptive systems that operate under dynamic workloads, select from alternative runtime tactics, express objectives as acceptable operating bounds over time, and can support a symbiotic simulation component seeded from live runtime observations — conditions under which runtime stability can be treated as an explicit decision criterion rather than a post-hoc evaluation property. Plausible extensions include IoT, microservice and edge computing environments, each requiring domain-specific instantiation of the tactic space, stability metrics and look-ahead horizon. Applicability is bounded by the assumption that a sufficiently faithful runtime model can be constructed and updated within the control interval, and that the three-interval look-ahead horizon suits the target system's workload dynamics and settling-time characteristics. Broader claims therefore require evaluation across additional domains and workload profiles.

Threats to Validity. The findings should be interpreted within the scope of the experimental design. The evaluation uses a single benchmark (*RUBiS*) and workload trace (*World Cup 1998*), compares against a single reactive baseline, and is confined to *CloudSim* simulation rather than an emulated or real deployment; simulation enables controlled, repeatable comparison but may not capture deployment effects such as network variability, hypervisor overhead, or provisioning latency. The *World Cup 1998* was selected for its extreme burstiness, useful for stress-testing, but may not reflect the diurnal, multi-tenant, or service-to-service patterns of modern cloud workloads; generalisability to such workloads has not been evaluated. Richer comparisons against utility-based or proactive controllers, together with validation across additional systems, workloads and real environments, are therefore required. Runtime stability is also a multi-dimensional construct, and the metrics and weighting scheme adopted here represent one instantiation among several, which may lead to different trade-offs; the sensitivity of the ranking function to alternative weight configurations was not evaluated. The symbiotic simulation further assumes constant workload throughout the look-ahead horizon, which may affect predicted behaviour for tactics evaluated near workload transitions. Finally, the computational cost of the symbiotic *what-if* evaluation was not measured independently, so the reported overhead reflects aggregate adaptation time rather than the isolated simulation cost.

VIII. Conclusion and Future Work

This paper presented a runtime stability-aware adaptation approach for self-adaptive systems, evaluated in the domain of cloud architectures using symbiotic simulation. Within an autonomic control loop, candidate tactics are evaluated before enactment according to their predicted effect on both service stability and adaptation quality, allowing runtime stability to be treated as an explicit system property in runtime control rather than as a post-hoc evaluation measure. Implemented as an extension of *CloudSim* and evaluated using the *RUBiS* benchmark with workloads derived from the *World Cup 1998* trace, the approach reduced operational cost, peak resource utilisation and adaptation frequency relative to a baseline reactive controller, whilst maintaining response time within the stability objective, achieving equivalent energy consumption, and reducing response-time variability across all service type configurations — a stabilisation effect beyond mean response-time improvement. These results show that runtime stability-aware decision making can improve the efficiency and predictability of self-adaptation outcomes in a controlled setting.

Future work will extend the evaluation using OpenStack and learning-based MAPE-K mechanisms, and through comparison against utility-based and proactive controllers, to establish how the contribution of stability-aware ranking compares with alternative tactic-selection strategies. Incorporating workload forecasting into the look-ahead horizon may improve adaptation frequency and resource overshoot estimates during volatile periods. A systematic sensitivity analysis over alternative weight configurations would clarify how far the reported improvements depend on the specific weighting used in this study.

ACKNOWLEDGMENT

M. Salama gratefully acknowledges Rodrigo Calheiros for his support in the development of the simulation environment and Patricia Lago for her guidance on architectural evaluation.

REFERENCES

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[2] B. Cheng and et al., *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer-Verlag, 2009, pp. 1–26.

[3] R. d. Lemos and et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2013, vol. 7475, pp. 1–32.

[4] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2009, pp. 1–11.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *ACM Communications*, vol. 53, no. 4, pp. 50–58, 2010.

[6] N. M. Villegas, H. A. Muller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2011, pp. 80–89.

[7] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[9] "Rice university bidding system." [Online]. Available: www.rubis.ow2.org

[10] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, 2000.

[11] R. Ghosh, F. Longo, V. K. Naik, and K. S. Trivedi, "Quantifying resiliency of IaaS cloud," in *29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 343–347.

[12] A. Pataricza, I. Kocsis, A. Salánki, and L. Gönczy, "Empirical assessment of resilience," in *Software Engineering for Resilient Systems*, ser. Lecture Notes in Computer Science, A. Gorbenko, A. Romanovsky, and V. Kharchenko, Eds. Springer Berlin Heidelberg, 2013, vol. 8166, pp. 1–16.

[13] A. Gorbenko, V. Kharchenko, O. Tarasyuk, Y. Chen, and A. Romanovsky, "The threat of uncertainty in service-oriented architecture," in *RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*. ACM, 2008, pp. 49–54.

[14] A. Gorbenko, V. Kharchenko, S. Mamutov, O. Tarasyuk, Y. Chen, and A. Romanovsky, "Real distribution of response time instability in service-oriented architecture," in *29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 92–99.

[15] R. Almeida and M. Vieira, "Changeloads for resilience benchmarking of self-adaptive systems: A risk-based approach," in *9th European Dependable Computing Conference (EDCC)*, 2012, pp. 173–184.

[16] S. Kounev, P. Reinecke, F. Brosig, J. T. Bradley, K. Joshi, V. Babka, A. Stefanek, and S. Gilmore, "Providing dependability and resilience in the cloud: Challenges and opportunities," in *Resilience Assessment and Evaluation of Computing Systems*, K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, Eds. Springer Berlin Heidelberg, 2012, pp. 65–81.

[17] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Research issues in symbiotic simulation," in *Proceedings of Winter Simulation Conference (WSC)*, 2009, pp. 1213–1222.

[18] S. J. Turner, "Symbiotic simulation and its application to complex adaptive systems (keynote)," in *Proceedings of IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2011, pp. 3–3.

[19] B. Tjahjono and J. Xu, "Linking symbiotic simulation to enterprise systems: Framework and applications," in *Proceedings of 2015 Winter Simulation Conference (WSC)*, 2015, pp. 823–834.

[20] Y. Abuseta and K. Swesi, "Towards a framework for testing and simulating self adaptive systems," in *6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2015, pp. 70–76.

[21] A. Elhabbash, M. Salama, R. Bahsoon, and P. Tino, "Self-awareness in software engineering: A systematic literature review," *ACM Trans. Auton. Adapt. Syst.*, vol. 14, no. 2, pp. 1–42, 2019.

[22] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "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)*. ACM, 2015, p. 1?12.

[23] J. Cámara, G. A. Moreno, D. Garlan, and B. Schmerl, "Analyzing latency-aware self-adaptation using stochastic games and simulations," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 4, 2016.

[24] J. Cámara, D. Garlan, G. A. Moreno, and B. Schmerl, "Analyzing self-adaptation via model checking of stochastic games," in *Software Engineering for Self-Adaptive Systems III. Assurances*, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer International Publishing, 2017, pp. 154–187.

[25] I. Brandic, "Towards self-manageable cloud services," in *Proceedings of 33rd Annual IEEE International Conference on Computer Software and Applications (COMPSAC)*, vol. 2, 2009, pp. 128–133.

[26] N. A. A. Gillam, "Intelligent techniques and architectures for autonomic clouds: introduction to the itaac special issue," *Journal of Cloud Computing*, vol. 1, no. 1, 2012.

[27] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for cloud infrastructure management," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 472–487, 2013.

[28] P. Jamshidi, A. Ahmad, and C. Pahl, "Autonomic resource provisioning for cloud-based software," in *Proceedings of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014.

[29] M. Salama, R. Bahsoon, and P. Lago, "Stability in software engineering: Survey of the state-of-the-art and research directions," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019, salama2019.

[30] P. Kathiravelu and L. Veiga, "Concurrent and distributed cloudsim simulations," in *IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2014, pp. 490–493.

[31] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *4th IEEE International Conference on Utility and Cloud Computing (UCC)*, 2011, pp. 105–113.

[32] A. Beloglazov and R. Buyya, "OpenStack neat: A framework for dynamic consolidation of virtual machines in OpenStack clouds: A blueprint," *Cloud Computing and Distributed Systems (CLOUDS) Laboratory*, 2012.

[33] Amazon Web Services, Inc., "Amazon EC2 Instance Types," accessed: 2017-10-01. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[34] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.