Mathematical model of the cloud infrastructure life cycle⋆

Sergii Lysenko[1,*,†], Oleh Bondaruk[1,†], Oleksii Bondar[1,†], Liudmyla Koretska[1,†], and Tomas Sochor[2,†]

[1] *Khmelnitsky National University, Khmelnitsky, Instytutska street 11, 29016, Ukraine*
[2] *European Research University, Ostrava, Czech Republic*

Abstract

This paper presents a formalized mathematical model for managing the life cycle of cloud infrastructure. The proposed model incorporates a multi-layered architecture that includes the infrastructure, cloud services management, and IT governance levels, capturing both automated and human-centric control processes. Using finite automata and process algebra, the model supports a structured representation of service states, control actions, and transition dynamics. It enables real-time lifecycle automation, SLA compliance enforcement, and adaptive response to dynamic workloads. The effectiveness of the model is validated through a series of experiments simulating real-world scenarios, demonstrating high service availability, resource optimization, and system scalability. This work contributes to the formalization and automation of cloud infrastructure lifecycle management, providing a foundation for further tool development and integration into cloud orchestration platforms.

Keywords

Cloud infrastructure lifecycle, mathematical modeling, finite automata, lifecycle automation, resource management, service level agreement, cloud service orchestration

1. Introduction

The rapid development of digital technologies and the increasing demand for scalable computing resources have led to the widespread adoption of cloud computing as a fundamental paradigm for delivering IT services [1-3]. Cloud infrastructure has become an integral component of modern information systems, providing on-demand access to computational power, storage, and networking capabilities [4-6]. However, the management of cloud infrastructure throughout its entire life cycle from design and deployment to scaling, maintenance, and decommissioning presents a complex challenge that requires systematic and formalized approaches [7-9].

Despite significant progress in cloud infrastructure automation and orchestration, there is still a lack of comprehensive mathematical models that accurately describe the dynamics of its life cycle. Such models are essential for predicting system behavior, optimizing resource allocation, ensuring reliability, and supporting decision-making in cloud infrastructure management [10-12]. Existing approaches often rely on empirical or heuristic methods, which may not fully capture the underlying processes or support rigorous analysis and verification [13-15].

This paper proposes a formal mathematical model of the cloud infrastructure life cycle, based on discrete-state representations and process algebra principles. The model reflects the sequential and parallel transitions between different infrastructure states, taking into account provisioning

ICyberPhyS'25: 2nd International Workshop on Intelligent & CyberPhysical Systems, July 04, 2025, Khmelnytskyi, Ukraine⋆
[1]* Corresponding author.
† These authors contributed equally.
sirogyk@ukr.net (S. Lysenko); oleg6467@ukr.net (O. Bondaruk); tineyalex@gmail.com (O. Bondar); koretskal@khmnu.edu.ua (L. Koretska); tomas.sochor@eruni.org (T. Sochor)
0000-0001-7243-8747 (S. Lysenko); 0009-0000-8663-4124 (O. Bondaruk); 0009-0003-6707-2981 (O. Bondar); 0000-0002-4284-4936 (L. Koretska); 0000-0002-1704-1883 (T. Sochor)

constraints, resource dependencies, and performance metrics. The proposed approach enables the analysis of system evolution over time and supports the development of tools for automated lifecycle management in cloud environments [16-18].

The remainder of this paper is structured as follows: Section II reviews related work in cloud infrastructure modeling. Section III presents the formal definition of the proposed model. Section IV provides case studies and evaluation results. Finally, Section V concludes the paper and outlines directions for future research.

2. Related works

Resource management is a core aspect of cloud computing and virtualization. It involves the allocation, coordination, release, and monitoring of cloud resources to ensure efficient and effective system performance. Due to the virtualized, heterogeneous, and multi-tenant nature of cloud environments, managing such resources is inherently complex. Challenges arise from uncertainty, diversity, large-scale infrastructure, and unpredictable workloads generated by numerous users. These factors hinder accurate global state estimation and workload forecasting, making resource management highly demanding. To address these issues, cloud resource management requires autonomous and adaptive strategies to optimize resource utilization while avoiding over- and under-provisioning.

Numerous centralized resource management approaches have been proposed in the literature, primarily adopting centralized architectures to support cloud applications.

The article [19] proposes a Petri net-based model for resource scheduling and auto-scaling in elastic cloud computing environments. The goal is to improve the allocation of virtual resources dynamically based on the workload demands. By modeling the system behavior using Colored Timed Petri Nets (CTPNs), the authors enable precise representation and simulation of complex cloud operations, including task arrival, resource provisioning, and scaling decisions. The model helps ensure optimal use of resources while maintaining service level agreements (SLAs). Performance evaluation shows that the proposed model supports efficient auto-scaling and task scheduling, enhancing system responsiveness and resource utilization.

Article [20] reviews and analyzes various policies and mechanisms aimed at improving resource management in cloud computing from a performance perspective. It highlights key challenges such as efficient resource allocation, load balancing, scalability, and energy efficiency. The authors explore existing strategies, including virtual machine (VM) migration, resource scheduling, and energy-aware management, and discuss their impact on system performance. The paper also identifies gaps in current research and suggests directions for developing more intelligent and adaptive resource management solutions to optimize cloud infrastructure performance.

The article [21] presents a proactive, self-managing framework for resource allocation in cloud computing environments. The framework uses autonomic computing principles to monitor, analyze, and manage cloud resources dynamically, aiming to improve performance, scalability, and resource utilization for service-based applications. By predicting workloads and adjusting resources in advance, the approach minimizes latency and service disruptions, contributing to more efficient and intelligent cloud infrastructure management.

Article [22] presents a cost-aware, elastic caching strategy for cloud environments using a Time-To-Live (TTL) based approach. The authors develop a mathematical model to dynamically adjust the cache size in response to workload changes, aiming to balance performance (hit rate) and operational cost. Their TTL-based mechanism is lightweight and adaptable, allowing cloud providers to provision cache resources elastically while keeping expenses under control. Through theoretical analysis and simulations, the proposed method demonstrates improved cost-efficiency and cache performance compared to traditional static or heuristic-based approaches.

The article [23] proposes a self-adaptive resource allocation approach for cloud-based software services using an iterative Quality of Service (QoS) prediction model. The method continuously monitors service performance and predicts future QoS values to dynamically adjust resource

allocation. It combines machine learning techniques with feedback control to respond to changing workloads and maintain SLA compliance. The proposed approach enhances resource efficiency and service reliability, as shown through experimental validation on real-world cloud service scenarios.

Article [24] provides a comprehensive review of cloud resource management techniques within the context of Industry 4.0, focusing on the integration of cloud computing with smart manufacturing and industrial automation. The authors categorize and evaluate various resource provisioning, scheduling, and optimization strategies, highlighting their relevance to real-time and data-intensive industrial applications. The paper also discusses key challenges, such as latency, scalability, energy efficiency, and security, that arise when deploying cloud solutions in Industry 4.0 environments. The review identifies gaps in current research and suggests future directions for intelligent, adaptive, and decentralized resource management systems.

The article [25] presents a hybrid resource allocation algorithm for cloud computing that combines the Shuffled Frog Leaping Algorithm (SFLA) and the Cuckoo Search (CS) algorithm. The proposed hybrid approach aims to enhance resource utilization, minimize execution time, and reduce energy consumption in cloud environments. By integrating the exploration capability of Cuckoo Search with the local search efficiency of SFLA, the algorithm achieves better optimization results than using either technique alone. Simulation results demonstrate that the hybrid method outperforms traditional approaches in terms of task completion time and resource efficiency.

Article [26] introduces an autonomic task scheduling algorithm designed to handle dynamic workloads in cloud computing environments using an effective load balancing technique. The proposed method enables the cloud system to automatically adapt to workload changes by distributing tasks efficiently across virtual machines (VMs), ensuring improved resource utilization, reduced response time, and enhanced system performance. The algorithm incorporates self-management features inspired by autonomic computing principles, allowing it to operate with minimal human intervention. Experimental results show that the approach significantly improves load distribution and task execution efficiency compared to traditional scheduling methods.

Article [27] presents a distributed edge computing framework enhanced with artificial intelligence (AI) to support Internet of Things (IoT) applications. The proposed solution focuses on decentralized decision-making and resource management at the edge of the network, aiming to reduce latency, improve scalability, and ensure efficient service delivery. AI techniques are integrated to enable smart task offloading, real-time adaptation, and intelligent data processing. The framework supports dynamic environments typical of IoT scenarios and demonstrates improved performance in terms of latency, energy efficiency, and computational load distribution.

Paper [28] presents a method for efficient resource provisioning in cloud systems by leveraging workload prediction techniques. The approach forecasts future resource demands using historical workload data, enabling proactive allocation of computing resources. By accurately predicting workload fluctuations, the system reduces resource wastage and improves overall cloud performance and cost efficiency. Experimental results validate the model's ability to enhance scalability and responsiveness compared to reactive provisioning methods.

Article [28] proposes a resource provisioning mechanism for cloud systems that leverages workload clustering combined with the Biogeography-Based Optimization (BBO) technique. The approach clusters workloads with similar resource demands to optimize resource allocation more effectively. BBO is employed to find optimal provisioning solutions, improving resource utilization and reducing operational costs. The method aims to enhance scalability and efficiency in dynamic cloud environments by adapting resource allocation based on workload patterns. Experimental results demonstrate improved performance over traditional provisioning strategies.

Analysis has shown that there is a strong need in construction of a formal model to represent the entire lifecycle of cloud infrastructure, encompassing the provisioning, operation, scaling, monitoring, and decommissioning phases of cloud services. The model has to address the key challenges in cloud lifecycle management, including dynamic scaling, SLA policy enforcement, and adaptive control in response to variable workloads and system events.

3. Cloud service lifecycle models

To solve the problem of increasing the efficiency of curation by the life cycle of cloud infrastructure, we will describe its mathematical model. It is obvious that it has a multi-level hierarchical organization of management and includes:

- Infrastructure layer (servers, networks, storage).
- Cloud Services Lifecycle Management Layer.
- IT management level (according to ITIL, with human intervention).

Let us consider a formalized mathematical model that takes into account these levels:

$$M_{cloud} =< R, S, M, H >, \quad (1)$$

where R is a set of resources (physical or virtual): servers, storages, network devices; $R = \{r_1, r_2, \dots, r_n\}$;

$S = \{s_1, s_2, \dots, s_m\}$, is a set of cloud services, each of which is an aggregated set of resources;

$M = \{m_1, m_2, \dots, m_k\}$ – a set of control processes of the life cycle (monitoring, scaling, recovery, billing, etc.);

$H = \{h_1, h_2, \dots, h_p\}$ – a set of IT management processes (according to ITIL: change, release, incident management).

The description of the functioning of the life cycle of a cloud service $s_i \in S$ will look like an automaton described as a finite automaton:

$$A_i = (Q, \Sigma, \delta, q_0, F), \quad (2)$$

where:

$Q = \{q_{init}, q_{provisioned}, q_{running}, q_{paused}, q_{scaled}, q_{failed}, q_{terminated}\}$ – set of service states;

Σ – set of events/triggers (deploy, scale, pause, resume, fail, recover, terminate);

$\delta: Q \times \Sigma \to Q$ – transition function;

$q_0 = q_{init}$ – initial state;

$F = \{q_{terminated}\}$ — final state.

Let us define a formal model that describes the behavior of the control system in the form of an automaton (state machine or other type), which defines the rules for controlling processes$m_j \in M$, objects, where each process is defined as a control mechanism superimposed on A_i.

Specifically, the system monitoring function, which returns the metric vector (load, response time, power consumption, etc.), will look like this:

$$monitor: S \times T \to R^d. \quad (3)$$

The scaling function is described by a rule of type:

$if\ monitor(s_i, t) \geq \theta \Rightarrow \Sigma \ni scale_{up} \Rightarrow \delta(q_{running}, scale\_up) = q_{scaled}$.

Let us consider the level of IT management, namely the process of human-centric control. Processes $h \in H$ affect the transitions of automata A_i, but indirectly through approval, approval or interaction with people. Let's formalize this through the function:

$\gamma: \Sigma \to \{true, false\}$,

where $\gamma(\sigma)$=true means that the event is allowed according to ITIL/organizational process policies.

For example, the deploy(s_i) event is only possible when:

$\gamma(deploy) = approval(h_k, s_i) = true$.

Let us consider the infrastructure layer, which operates with a set of resources R, organized in ensembles – managed resource pools.

Let's assume $E = \{e_1, e_2, \dots, e_l\} \subseteq P(R)$ as a set of ensembles, where each $e_i \subseteq R$ is a pool of homogeneous resources.

Then we will describe each ensemble by capacity $C(e_i) = \{c_1, c_2, \dots, c_d\}$ (CPU, memory, network); with a distribution strategy $\phi: S \to E$ that assigns services to ensembles; as well as rules of self-organization (autonomous management).

Let us describe the general compositional model by combining all levels.

Each service is linked to resources through $s_i \in S\varphi$ mapping, controlled through an automaton, which is influenced A_i by M automated control services and H manual (ITIL) processes.

The conditions for the correct functioning of the service will be set by the rule:

$$\forall s_i: \exists e_j: \phi(s_i) = e_j \wedge resources(e_j) \geq demand(s_i)\,. \quad (4)$$

Then the integral function of the state of the system will look like:

$$S(t) = (\,\cup_{i=1}^{m} A_i(t)\,, \cup_{j=1}^{k} m_j(t)\,, \cup_{h=1}^{p} h_j(t)\,, \cup_{l=1}^{n} r_i(t)\,). \quad (5)$$

3.1. Cloud service model

Let's describe a mathematical model of a cloud service that takes into account the aggregation of resources into higher-level services, the life cycle states of a cloud service, the hierarchy of service types (IaaS, PaaS, SaaS), as well as management automation based on virtualization and resource pools.

Let's take as a set $R = \{r_1, r_2, \dots, r_n\}$of physical or virtual resources (CPU, RAM, storage, network adapters, etc.), but as a set $V = \{v_1, v_2, \dots, v_m\}$of virtualized resources derived from R using virtualization technologies ν, where $\nu: P(R) \to P(V)$ is a virtualization function.

This means that real resources are aggregated into pools of virtualized resources, which later become elements of cloud services.

Next, let's define a CS cloud service defined as a tuple:

$$CS_i = \langle ID_i, T_i, V_i, L_i, SLA_i, S_i(t) \rangle\,, \quad (6)$$

where:

IDi – unique identifier of the service;

$T_i \in \{IaaS, PaaS, SaaS\}$ – type of service (infrastructure, platform, software);

$V_i \subseteq V$ – a set of virtualized resources that make up the service;

L_i – service life period;

SLA_i – a set of requirements for service (Service Level Agreement);

$S_i(t)$ – the state of the service in time (determined by the life cycle).

The life cycle of a cloud service is presented in the form of a finite automaton:

$$A_i = (Q, \Sigma, \delta, q_0, F), \quad (7)$$

where:

$Q = \{q_{requested}, q_{provisioned}, q_{running}, q_{paused}, q_{rescaled}, q_{terminated}\}$ – life cycle states,

Σ – events (for example: request, provision, start, pause, scale, terminate),

$\delta: Q \times \Sigma \to Q$ is the function of transition between states,

$q_0 = q_{requested}$ – initial state,

$F = \{q_{terminated}\}$ – final state.

Let's define the transitions as follows:

$$\delta(q_{requested}, provision) = q_{provisioned}$$
$$\delta(q_{provisioned}, start) = q_{running}$$
$$\delta(q_{running}, pause) = q_{paused}$$
$$\delta(q_{running}, scale) = q_{rescaled}$$
$$\delta(\cdot, terminate) = q_{terminated}$$

To reduce the complexity of administration, resources are aggregated into autonomous pools:

$$P = \{p_1, p_2, \dots, p_k\}, p_j \subseteq R$$

Each pool hides the hardware implementation and is managed using $p_j$self-organization policies. Then let's define the aggregation function as $\alpha: P(R) \to P$, and the assignment function $\phi: CS \to P$, which determines which resource pool supports a particular service.

The function of the composition of services is as follows:

$$\psi: P(CS_I) \to CS_P,\ \psi': P(CS_P) \to CS_S\ . \tag{8}$$

Let's consider the process of automating event processing by a cloud service. Automation is implemented through event functions, where the query function will look like:

$$req - uest: U \times T \times params \to \Sigma\ , \tag{9}$$

where U user, params – service parameters (CPU, RAM, lifetime, etc.).

Then we will present the function of reaction to events as follows:

$$\rho: \Sigma \times CS \to A_i\ , \tag{10}$$

Description of the model of functioning of a cloud service includes determining the time limit of its existence. Let's assume t_{start} – the time of activation of the service, . Then for the time $t_{end} = t_{start} + \Delta t$, де $\Delta t \in L_i$t the service is active if:

$$t \in [t_{start}, t_{end}] \Rightarrow S_i(t) \neq q_{terminated}. \tag{11}$$

After the time has elapsed, the service is automatically destroyed, and the resources are returned to the pool:

$$S_i(t > t_{end}) = q_{terminated}, V_i \to P. \tag{12}$$

3.2. Cloud service lifecycle model

Based on the above description of the life cycle of a cloud service, it is possible to build a formalized mathematical model that describes the stages, parameters, actions and states of the service during its existence. This model is based on the concept of a finite automaton with state preservation, a parameterized transition graph and a controlled set of control operations.

Let's denote a cloud service as an object S, which is defined by:

$$S = \langle T, O, I, M, \Sigma, \delta, s_0, s_f \rangle, \tag{13}$$

where:

T is the set of service templates;

O is the set of specific service offers;

I is the set of service instances

M is the set of control operations;

Σ is the set of events (triggers) of the life cycle;

$\delta: S \times \Sigma \to S$– the function of transitions between states;

s_0 – initial state (pattern definition);

s_f – final state (termination of the service).

Then the life cycle of a cloud service can be described as a set of states:

$$L = \{s_0, s_1, s_2, s_3, s_4, s_f\} \tag{14}$$

where:

s_0 – creating a T service template,

s_1 – creating a T-based sentence O,

s_2 – creating an instance of service I,

s_3 – active use of I,

s_4 – performing operations M,

s_f – completion, return of resources.

Each instance is determined by the following parameters:$i \in I$:

$$i = \langle C, A, P, D, SLA \rangle, \tag{15}$$

where:

C – computing capacity (CPU, RAM);

A – availability requirements;

P – performance;

D is the duration of the lease;

SLAs – Service Level Agreements.

The transition function that displays the reaction to events will look like:

$$\delta(s, \sigma) = s' . \tag{16}$$

For the main events $\sigma \in \Sigma$, we determine the set of events that are given in Table 1.

Table 1

Set of cloud service events

Event (σ)	Initial state	New state
create_template	-	s_0
publish_offering	s_0	s_1
subscribe_request	s_1	s_2
instantiate_success	s_2	s_3
manual_or_autonomic_management	s_3	s_4
terminate_request or timeout	s_3, s_4	s_f

Let's specify a set of control operations:

$M = \{scale, backup, start, stop, terminate, monitor\}$,

where each operation is a function:

$$m: I \times P \to I' , \tag{17}$$

where P is the parameters for calling the operation, and I′ is the new state of the cloud service instance.

Let's describe the process of autonomous driving. Let's take $\mu: I \times SLA \to M \times P$ as a function a recommendation or automatic execution, where $\mu(i, SLA_i)$ returns which control operation should be performed and with what parameters if a deviation from the SLA is detected.

The service template is described by a tuple of the form:

$T = \langle Topology, BuildPlan, \{m_i\}_{i=1}^{n} \rangle$,

$Topology$ - a graph of components and relationships between them,

$BuildPlan$ – a sequence of steps to create an instance,

$m_i \in M$ – management operations.

Let be the resources allocated to instance $R = \{r_1, r_2, \ldots, r_k\}$.

In case of completion $i \to s_f$, the following rule is used:

$$\forall r_j \in R_i, \; r_j \to Pool, \tag{18}$$

Final state graph can be presented as:

[s0] ⟶create_template⟶ [s0]

[s0] ⟶publish_offering⟶ [s1]

[s1] ⟶subscribe⟶ [s2]

[s2] ⟶instantiate_success⟶ [s3]

[s3] →management_op→ [s4]
[s4] →monitoring/trigger→ [s3]
[s3|s4] →terminate/timeout→ [sf]

3.3 Formalization of requirements for automation of lifecycle management of cloud environments

Automation of the management of the life cycle of a cloud environment is a key factor in ensuring its efficiency, reliability, scalability, and continuity. Unlike traditional IT systems, cloud services operate in a dynamic, multi-user environment with a high degree of virtualization and variability of configurations, which necessitates a clear formalized approach to defining automation requirements.

The purpose of requirements formalization is to provide a unified model for managing cloud resources and services, the ability to machine interpret component life cycles, support adaptive management and autonomous response to events, and ensure consistency between SLA policy parameters, instance configurations, and management plans.

Automation should provide a response to internal and external events in real time:

- start/stop the service;
- load change;
- violation of the service level management policy;
- depletion of resources;
- changes in security policies [30, 31].

Let's formalize the process of automating the management of the life cycle of the cloud environment:

$$\forall \sigma \in \Sigma, \exists m \in M: \mu(\sigma) = m(P), \tag{19}$$

where Σ is the set of events, M is the set of possible actions, μ is the function of the correspondence of events to actions, P is the execution parameters.

The configuration of the cloud environment is defined in the form of formalized templates or specifications:

$$T = \langle C, D, R \rangle , \tag{20}$$

where:

C – a set of components (servers, storages, networks),

D is the dependencies between them (graph or tree),

R – restrictions and rules (security, availability, scaling policies) [32, 33].

We will understand that templates support machine interpretation and automatic deployment.

Then management must take into account the state of the service $s_i \epsilon S$, where $S = \{s_0, s_1, \dots, s_f\}$ is the set of states (definition, publishing, instantiating, operating, termination); $\delta: S \times \Sigma \to S$ is an event-based state transition function.

Thus, such formalization allows you to automate:

activation/deactivation of services;

change of management plans depending on the state;

control of the end of the life cycle.

The system has the ability to automatically scale based on monitoring data:

$$m_{scale}: I \times \{CPU, RAM, net, \dots\} \to I' . \tag{21}$$

The execution condition will look like this:

$$\exists \theta: if\ M(x) > \theta \Rightarrow m_{scale}, \tag{22}$$

where M(x) is the value of the load metric, θ is the threshold value.

Let us consider the process of ensuring the policy of service level management. Support and automatic control of the implementation of service level agreements (SLAs) consist in monitoring critical parameters, automatic notification of violations, and applying corrective actions.

Formally, the process of ensuring the service level management policy will be described as:

$$SLA = \{(m_i, \omega_i, \tau_i)\}_{i=1}^{n}, \quad (23)$$

where:

m_i is the metric;

ω_i is the allowable range;

τ_i is the time interval of te check.

Let's consider the process of automated logging and auditing. Each action should generate a log entry: who initiated the action; when; to which resource it is applied; The result of the action.

Formally, the process of automated logging and auditing will be presented as follows:

$$L = \{(t_i, u_i, a_i, r_i, s_i)\}_{i=1}^{n}, \quad (24)$$

where – time, – user or service, – action, – resource, – execution status.$t_i u_i a_i r_i s_i$

Let's present a functional model of automated control.

Formalized as a system is described by a tuple:

$$A = \langle I, M, \Sigma, \delta, \mu, SLA, T \rangle , \quad (25)$$

where:

I is the set of service instances;

M – controlled actions;

Σ – events;

δ is the function of transitions between states;

μ – strategy for responding to events;

SLAs – Service Level Policies;

T – configuration templates.

4. Experiments

System stability in this context can be interpreted as the ability of a cloud service to operate for a long time without noticeable failures or drops in performance, even under variable loads or unforeseen circumstances. Stability can be assessed by several main criteria:

To assess the stability of each stage of the cloud service lifecycle, it is important to analyze the transitions between different states (for example, from the " Active " state to the " Error " or "Scaling" state). Key factors that can affect stability:

- Load.
- Hardware problems.
- Interference between services.
- Monitoring and error handling.

Peak load analysis (for example, autoscaling can cause overload during the scaling process).

Detecting errors or malfunctions in hardware components.

Multiple services can interact with each other through shared resources (e.g., network, storage).

The system must be able to detect errors at each stage and transition to recovery mode without major delays or performance degradation.

An important part of stability is the ability to perform operations autonomously at each stage. For example, autoscaling should work without human intervention, ensuring that a stable state is maintained under changing conditions. Statistical analysis of automatic actions (for example, how

long it takes to scale or restore) helps to verify whether the system can automatically adapt to new conditions.

Detecting and analyzing responses to critical events, such as errors (the " Error " state) or high loads, can provide insight into the system's ability to recover or restore service stability.

4.1. Analysis of the effectiveness of the model

System effectiveness is determined by the ability of a cloud service to meet user requirements and adhere to service level management policies, ensuring high availability and performance with minimal resource consumption.

To assess the effectiveness of the model, you can measure:

- Service response time under different loads.
- System performance at different stages of its lifecycle, such as activation, scaling, or recovery.
- Resource usage (CPU, memory, network resources) at each stage.
- Time taken to perform operations: How quickly the system transitions between states (e.g., time from startup to activation or scaling time).

4.2. Resource optimization

Resource allocation analysis is an important step in performance analysis. The assessment of indicators such as:

- assessing the effective use of memory, processor time, disk space, and network.
- checking how effectively resources are scaled when the load changes.
- assessment of energy consumption during the implementation of various stages of the life cycle, especially during the scaling and recovery stages.

The key point is compliance with the service level management policy. The effectiveness of the model is assessed through service availability (analysis of the compliance of the service uptime with the service level management policy requirements), response time performance (analysis of the compliance of the service response time with the requirements), latency and recovery time (analysis of the system recovery time after failures or malfunctions).

Efficiency also includes optimizing resource costs, including infrastructure costs (analysis of the cost of deployment, scaling, support, and service closure), as well as resource utilization optimization (analysis of optimal resource utilization at each stage of the lifecycle).

4.3. Analysis of the stability and efficiency of the functioning of the cloud service life cycle model. Experimental setup

Analysis of the stability and efficiency of the cloud service lifecycle model is a comprehensive process that includes assessing performance, resource efficiency, compliance with service level management policies, and ensuring uninterrupted system operation. The use of mathematical modeling, monitoring, and optimization methods allows you to obtain accurate data to ensure high quality user service in the cloud environment.

To analyze the stability and efficiency of the cloud service lifecycle model, a series of experiments was performed, including various stages of the service lifecycle (definition , offering, subscription and instantiation , production process, scaling, and termination).

The experiment was based on an analysis of the stability of the cloud service during the execution of various stages of the life cycle, an assessment of the effectiveness of resource management, as well as the implementation of service level management policies and compliance with parameters.

The list of hypotheses of the experiment is given in Table 2.

Table 2
List of experimental hypotheses

No.	Description
Hypothesis 1	A cloud service with properly configured scaling and resource management processes continues to function stably, even with changing load or minor failures at the infrastructure level.
Hypothesis 2	Low resource loss during transitions between different system states (e.g. from " Active " to " Scaling ") while adhering to service level management policies
Hypothesis 3	System efficiency (compliance with service level management policies and request processing speed) increases under conditions of dynamic scaling and adaptive load management

The purpose of the experiment is to test whether the stability and efficiency conditions of the cloud service are met within the life cycle model. Particular attention is paid to stability during scaling and under high loads, resource efficiency (CPU, memory, network usage), implementation of service level management policies (response time, service availability), as well as the costs of supporting and maintaining the service.

The experiment was conducted in a test cloud environment that matches the characteristics of a real cloud provider. The environment included:

1. Virtual machines for running applications.
2. Virtual storage resources and network resources.
3. Automated systems for scaling and load management.
4. Monitoring tools to collect data on load, resource usage, and service level management policy enforcement.

The experimental procedure included the following steps:

1. Initialization, which involved choosing a cloud service type (e.g., big data processing service, web hosting).
2. Stability testing, in which different loads were imposed on the system (for example, different numbers of simultaneous requests, resource loads), and an assessment of the system's time response to the load was also carried out, which included an analysis of transient processes (scaling, startup, shutdown).
3. Performance testing, which included estimating scaling time when the load changes, estimating resource costs (CPU, memory, network) during the execution of various stages of the lifecycle, and estimating recovery time after an error or overload.
4. Analysis of the implementation of service level management policies, which included an assessment of service availability and response time to requests, as well as an analysis of the implementation of service level management policies under high load conditions.
5. Scaling and adaptability, which included testing dynamic scaling under varying loads, as well as evaluating performance during transitions between different stages (scaling, recovery).

4.4. Experimental results

Let us consider the results of the system stability.

Under load, we will have the number of simultaneous requests coming to the cloud service.

Response time refers to the average time a service takes to respond to a request.

Execution time refers to the time it takes to process a request and response.

CPU usage corresponds to the average percentage of CPU usage during the test.

Memory usage indicates the average memory usage of the service.

Availability reflects the percentage of time during which the service was available and operating without failures.

Recovery time refers to the time it takes for a system to recover from a failure or overload.

Lost Requests displays the percentage of requests that were not processed due to excessive load or failures .

Scaling time refers to the time it takes to add or remove resources to support high-load operations.

Disaster recovery time refers to the time it takes for a system to recover from failures (such as a virtual machine crash).

As a result of the experiments, it was recorded that the system response time remained within acceptable values even under significant loads (within ±5% of the initial value).

The bandwidth was sufficient to handle 100% of requests without significant delays.

The scaling process was automatic and without significant delays, with a recovery time after scaling within 1–3 minutes.

In the event of a minor failure (for example, a virtual machine crash), the system was successfully restored within 5 minutes.

Resource usage during operations was within optimal values, without resource overload. CPU, memory, and network resources were used within 80–90% of maximum values.

The execution time of requests to the service remained stable even under high loads, with an average response time of 200-300 ms when processing 100 requests simultaneously.

The system efficiently handled high peak loads (up to 500 simultaneous requests), with only a slight (5-10%) decrease in throughput.

The results of experiments on stability and performance when executing queries are presented in Table 3.

The results of high load stability and recovery time are presented in Table 4.

Table 3
Stability and performance when executing queries

Experiment No.	Load (number of simultaneous requests)	Response time (ms)	Execution time (ms)	CPU usage (%)	Memory usage (%)	Availability (%)
1	50	180	250	75	60	99.98
2	100	200	300	80	65	99.95
3	200	250	350	85	70	99.90
4	300	280	400	88	75	99.93
5	500	320	450	90	80	99.92

Table 4
High load stability and recovery time

Test number	Load (number of simultaneous requests)	Recovery time (m)	Lost requests (%)	Scaling time (ms)	Recovery time after failure (ms)
1	50	3	0.1	120	300
2	100	4	0.3	150	350
3	200	5	0.5	180	400
4	300	6	0.8	210	450
5	500	7	1.0	250	500

Let's consider the results of experiments on the implementation of service level management (SLA) policies.

Response time is the average time it takes for a service to respond to a request.

Availability reflects the percentage of time during which the service is available and operating without interruption.

Service Level Management Policy Compliance reflects the percentage of service level agreement fulfillment, where higher values indicate greater compliance with the terms of the service level agreements.

Disaster recovery time: Recovery time from system failures, availability disruptions, or poor performance.

The service availability time was over 99.95%, which exceeds the standard service level management policy requirements for cloud services.

Recovery time after failures did not exceed 3 minutes, which meets the requirements of the service level management policy for critical services.

Response times were stable, ensuring that the service level management policy of 1 second for 90% of requests was met.

The results of high-load stability and recovery time are presented in Table 5.

Table 5

Results of experiments on the implementation of service level management policies

Test number	Response time (ms)	Availability (%)	SLA Compliance (%)	Recovery time after failures (ms)
1	180	99.98	100%	300
2	200	99.95	99.9%	350
3	250	99.90	99.8%	400
4	280	99.93	99.7%	450
5	320	99.92	99.5%	500

Let's consider the results of experiments on scaling and adaptability.

We will consider the change in load to be the percentage increase in the load on the system during the test.

Scaling time refers to the time it takes for the system to scale when the load changes.

Responsiveness defines the improvement in throughput after autoscaling, expressed as a percentage.

The load reduction time reflects the time it takes for the load to decrease after a peak in requests, when the load returns to normal levels again.

When the load increased, the system automatically scaled within 1-3 minutes, depending on the type of resource being scaled (for example, adding virtual machines or expanding memory).

Adaptive control was implemented in the system, which allowed the scaling strategy to be dynamically changed depending on the actual load.

The results of the scaling and adaptability experiments are presented in Table 6.

The results of the experiments showed that the tested cloud service demonstrated high stability during normal operation, and also effectively recovered after minor failures.

The system was able to efficiently use resources, ensuring stable service operation under high loads.

All key aspects of the service level management policy, including availability, response time, and recovery, were performed at or above standard requirements.

Automatic scaling and adaptive resource management provide high efficiency and reduced infrastructure costs, which is important for cloud services with dynamic loads.

These results indicate a high level of stability and efficiency of the proposed cloud service lifecycle model, making it suitable for use in real-world conditions.

Table 6
Scalability and adaptability

Test number	Load change (%)	Scaling time (ms)	Adaptability (throughput improvement) (%)	Load reduction time after peak (ms)
1	50%	120	25%	150
2	100%	150	30%	180
3	200%	180	40%	210
4	300%	200	45%	240
5	500%	250	50%	300

Conclusion

In this work, we developed a comprehensive and formal mathematical model to represent the entire lifecycle of cloud infrastructure, encompassing the provisioning, operation, scaling, monitoring, and decommissioning phases of cloud services. The model integrates discrete-state automata, virtualization abstractions, and hierarchical management levels, from infrastructure to ITIL-based manual controls, thereby offering a unified framework for understanding and automating the behavior of cloud systems.

Through the use of formal methods, the proposed model addresses key challenges in cloud lifecycle management, including dynamic scaling, SLA policy enforcement, and adaptive control in response to variable workloads and system events. The integration of monitoring functions, SLA validation mechanisms, and automated decision-making capabilities ensures not only the operational reliability of cloud services but also their compliance with business-level requirements.

The experimental evaluation confirms that the model supports high system stability and efficiency under real-world conditions. Specifically, it demonstrated the capability to maintain over 99.9% availability, handle up to 500 concurrent requests with minimal latency variation, and perform automated recovery and scaling within strict time constraints. These results underline the potential of the model to reduce resource waste, enhance service responsiveness, and minimize manual intervention, especially during peak load or failure events.

Additionally, the model facilitates fine-grained performance analysis, such as resource consumption tracking, SLA compliance measurement, and reaction time monitoring for lifecycle transitions. This enables cloud administrators to make data-driven decisions, optimize infrastructure costs, and improve the quality of service provided to end users.

Future work may focus on integrating this model into active orchestration tools, extending it with predictive analytics for preemptive scaling, and applying it to hybrid and edge-cloud environments. Moreover, further refinement of the SLA-driven automation policies can lead to even more resilient and self-adaptive cloud platforms.

In conclusion, the proposed formal model represents a significant step toward the systematic and automated management of cloud infrastructure lifecycles, paving the way for smarter, more efficient, and resilient cloud computing systems.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: grammar and spelling check; DeepL Translate in order to: some phrases translation into English. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] G. E. Goncalves, P. T. Endo, M. Rodrigues, D. H. Sadok, J. Kelner, C. Curescu, Resource allocation based on redundancy models for high availability cloud, *Computing* 102 (2020) 43–63.

[2] A. Hajisami, T. X. Tran, A. Younis, D. Pompili, Elastic resource provisioning for increased energy efficiency and resource utilization in cloud-RANs, Computer Networks 172 (2020) 107170.

[3] R. B. Halima, S. Kallel, W. Gaaloul, Z. Maamar, M. Jmaiel, Toward a correct and optimal time-aware cloud resource allocation to business processes, Future Generation Computer Systems 112 (2020) 751–766.

[4] I. Hamzaoui, B. Duthil, V. Courboulay, H. Medromi, A survey on the current challenges of energy-efficient cloud resources management, SN Computer Science 1 (2020) 1–28.

[5] H. O. Hassan, S. Azizi, M. Shojafar, Priority, network and energy-aware placement of IoT-based application services in fog-cloud environments, IET Communications 14 (2020) 2117–2129.

[6] Y. Hu, H. Zhou, C. de Laat, Z. Zhao, Concurrent container scheduling on heterogeneous clusters with multi-resource constraints, Future Generation Computer Systems 102 (2020) 562–573.

[7] J. Kumar, A. K. Singh, Decomposition based cloud resource demand prediction using extreme learning machines, Journal of Network and Systems Management 28 (2020) 1775–1793.

[8] H. Li, Y. Zhao, S. Fang, CSL-driven and energy-efficient resource scheduling in cloud data center, The Journal of Supercomputing 76 (2020) 481–498.

[9] M. Liaqat, V. Chang, A. Gani, S. H. Ab Hamid, M. Toseef, U. Shoaib, R. L. Ali, Federated cloud resource management: Review and discussion, *Journal of Network and Computer Applications* 77 (2017) 87–105.

[10] P. Abrol, S. Gupta, Social spider foraging-based optimal resource management approach for future cloud, The Journal of Supercomputing 76 (2020) 1880–1902.

[11] N. Gholipour, E. Arianyan, R. Buyya, A novel energy-aware resource management technique using joint VM and container consolidation approach for green computing in cloud data centers, Simulation Modelling Practice and Theory 104 (2020) 102127.

[12] S. S. Gill, S. Tuli, A. N. Toosi, F. Cuadrado, P. Garraghan, R. Bahsoon, H. Lutfiyya, R. Sakellariou, O. Rana, S. Dustdar, R. Buyya, ThermoSim: Deep learning-based framework for modeling and simulation of thermal-aware resource management for cloud computing environments, Journal of Systems and Software 166 (2020) 110596.

[13] P. Abrol, S. Guupta, S. Singh, Nature-inspired metaheuristics in cloud: A review, in: ICT Systems and Sustainability, Springer, Singapore, 2020, pp. 13–34.

[14] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, G. Fortino, Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA, IEEE Access 8 (2020) 54074–54084.

[15] P. A. Apostolopoulos, E. E. Tsiropoulou, S. Papavassiliou, Risk-aware data offloading in multi-server multi-access edge computing environment, IEEE/ACM Transactions on Networking 28 (2020) 1405–1418.

[16] P. A. Apostolopoulos, E. E. Tsiropoulou, S. Papavassiliou, Cognitive data offloading in mobile edge computing for internet of things, IEEE Access 8 (2020) 55736–55749.

[17] O. Ascigil, A. Tasiopoulos, T. K. Phan, V. Sourlas, I. Psaras, G. Pavlou, Resource provisioning and allocation in function-as-a-service edge-clouds, IEEE Transactions on Services Computing 1374 (2021) 1–14.

[18] A. Asghari, M. K. Sohrabi, F. Yaghmaee, A cloud resource management framework for multiple online scientific workflows using cooperative reinforcement learning agents, Computer Networks 179 (2020) 107340.

[19] K. R. Babu, P. Samuel, Petri net model for resource scheduling with auto scaling in elastic cloud, International Journal of Networking and Virtual Organisations 22 (2020) 462–477.

[20] M. Bansal, S. K. Malik, S. K. Dhurandher, I. Woungang, Policies and mechanisms for enhancing the resource management in cloud computing: A performance perspective, International Journal of Grid and Utility Computing 11 (2020) 345–366.
[21] T. Bhardwaj, H. Upadhyay, S. C. Sharma, An autonomic resource allocation framework for service-based cloud applications: A proactive approach, in: M. Pant, T. K. Sharma, R. Arya, B. C. Sahana, H. Zolfagharinia (Eds.), Soft Computing: Theories and Applications, vol. 1154, Springer, 2020, pp. 1045–1058.
[22] D. Carra, G. Neglia, P. Michiardi, Elastic provisioning of cloud caches: A cost-aware TTL approach, IEEE/ACM Transactions on Networking 28 (2020) 1283–1296.
[23] X. Chen, H. Wang, Y. Ma, X. Zheng, L. Guo, Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model, Future Generation Computer Systems 105 (2020) 287–296.
[24] B. K. Dewangan, A. Agarwal, T. Choudhury, A. Pasricha, S. Chandra Satapathy, Extensive review of cloud resource management techniques in industry 4.0: Issue and challenges, Software: Practice and Experience (2020) 1–20.
[25] P. Durgadevi, S. Srinivasan, Resource allocation in cloud computing using SFLA and Cuckoo search hybridization, International Journal of Parallel Programming 48 (2020) 549–565.
[26] F. Ebadifard, S. M. Babamir, Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment, Cluster Computing (2020) 1–27.
[27] G. Fragkos, E. E. Tsiropoulou, S. Papavassiliou, Artificial intelligence enabled distributed edge computing for Internet of Things applications, in: Proceedings of the 2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2020, pp. 450–457.
[28] L. J. Gadhavi, M. D. Bhavsar, Efficient resource provisioning through workload prediction in the cloud system, in: Smart Trends in Computing and Communications, Springer, Singapore, 2020, pp. 317–325.
[29] M. Ghobaei-Arani, A workload clustering based resource provisioning mechanism using biogeography based optimization technique in the cloud based systems, Soft Computing 25 (2020) 3813–3830.
[30] S. Lysenko, O. Savenko, K. Bobrovnikova, A. Kryshchuk, B. Savenko, Information technology for botnets detection based on their behaviour in the corporate area network, Communications in Computer and Information Science 718 (2017) 166–181.
[31] G. Markowsky, O. Savenko, S. Lysenko, A. Nicheporuk, The technique for metamorphic viruses' detection based on its obfuscation features analysis, CEUR Workshop Proceedings 2104 (2018) 680–687.
[32] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, K. Bobrovnikova, A technique for the botnet detection based on DNS-traffic analysis, Communications in Computer and Information Science 522 (2015) 127–138.
[33] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, K. Bobrovnikova, Anti-evasion technique for the botnets detection based on the passive DNS monitoring and active DNS probing, Communications in Computer and Information Science 608 (2016) 83–95.