Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Reliability-aware failure recovery for cloud computing based automatic train supervision systems in urban rail transit using deep reinforcement learning

Li Zhu[1], Qingheng Zhuang[1], Hailin Jiang[1*], Hao Liang[1], Xinjun Gao[2] and Wei Wang[3]

Abstract

As urban rail transit construction advances with information technology, modernization, information, and intelligence have become the direction of development. A growing number of cloud platforms are being developed for transit in urban areas. However, the increasing scale of urban rail cloud platforms, coupled with the deployment of urban rail safety applications on the cloud platform, present a huge challenge to cloud reliability.One of the key components of urban rail transit cloud platforms is Automatic Train Supervision (ATS). The failure of the ATS cloud service would result in less punctual trains and decreased traffic efficiency, making it essential to research fault tolerance methods based on cloud computing to improve the reliability of ATS cloud services. This paper proposes a proactive, reliability-aware failure recovery method for ATS cloud services based on reinforcement learning. We formulate the problem of penalty error decision and resource-efficient optimization using the advanced actor-critic (A2C) algorithm. To maintain the freshness of the information, we use Age of Information (AoI) to train the agent, and construct the agent using Long Short-Term Memory (LSTM) to improve its sensitivity to fault events. Simulation results demonstrate that our proposed approach, LSTM-A2C, can effectively identify and correct faults in ATS cloud services, improving service reliability.

Keywords ATS, Cloud computing, Urban rail transit, Reliability, Failure recovery

Introduction

China's economy has experienced rapid development driven by a new round of technological revolution and industrial transformation, leading to a glorious period of information construction in urban rail transit [1–4] . To achieve the unified deployment of urban rail applications, it is essential to construct an autonomous, controllable, and sustainable urban rail transit cloud platform [5]. This platform can break down the information barriers between subsystems and build an intelligent operation and maintenance system [6]. Urban rail transit clouds have been built and used in Hohhot, Wuhan, and other places. However, the continuous development in cloud applications [7] has resulted in increasingly complex cloud platform structures. Moreover, rail transit safety applications applied to cloud platforms are the future trend, as demonstrated by the implementation of cloud-based security computing platforms by companies such as Thales and Siemens. Therefore, a higher level of reliability is expected for the urban rail cloud.

*Correspondence:
Hailin Jiang
lhjiang@bjtu.edu.cn
[1] State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing, China
[2] Signal and Communication Research Institute of China Academy of Railway Sciences Group Co., Ltd., Beijing 100081, China
[3] Traffic Control Technology Co., Ltd., Beijing, China

The signaling system is at the core of the urban rail transit system, responsible for ensuring safe vehicle operation and improving driving efficiency. Automatic Train Supervision (ATS) is the primary component of the signaling system, and it will be deployed on the cloud platform. ATS is responsible for monitoring the on-time operation of trains [8]. The ATS failure will result in the inability of the mainline trains to receive timetable data from the central ATS system server, causing the loss of central ATS timetable functionality. The ATS failure will result in the inability of mainline trains to receive timetable data from the central ATS system server, leading to the loss of central ATS timetable functionality. Consequently, trains are unable to organize their operations according to the schedule, and the centralized control of the train operation organization mode cannot be implemented. Additionally, the central display screen fails to show train dynamics, and the interlocking of safety doors with train doors is disrupted. Events such as train delays occur, significantly impacting train operational efficiency.

Various methods are used to improve the reliability of cloud platforms, including fault removal [9], fault forecast [10, 11], and fault tolerance [12, 13]. Fault removal typically involves software-based detection and removal of potential faults in cloud systems. However, for complex systems, it is challenging to fully discover all potential faults. Fault forecast relies on accurately forecasting fault occurrences and employing preventive mechanisms based on prediction outcomes. In cloud computing, virtual machine migration is predominantly utilized to ensure service reliability. Among these methods, fault tolerance is the most widely used. It refers to the system's ability to perform its function correctly in the event of a failure [14, 15]. Fault tolerance is an essential requirement in cloud computing, achieved by employing redundancy configurations to enhance system reliability. Several studies have focused on fault tolerance in cloud computing, including the VM coordinated approach to detect deteriorating physical machines in data centers using Proactive Coordinated Fault Tolerant (PCFT) by the author of [16], the SVM-Grid based online fault detection approach proposed by Zhang et al. [17] to improve cloud stability, and the OPVMP model presented by Wang et al. [18] which uses a replication-driven method to improve the reliability of server-based cloud services. In [19], authors adopts traditional active-passive redundancy, providing backup instances for each node to facilitate recovery in case of failure. In [20], an active node deployment approach is proposed, employing a two-phase process: predicting traffic demands for each service chain and deploying instances using virtual machines. However, these methods fail to consider fault recovery mechanisms in the distributed environment of cloud computing. Nevertheless, in other distributed computing scenarios, researchers have proposed various methods to enhance link reliability. In [21], authors present AI-based trust management method to secure clustering to reliable and real-time communications. In [22], author propose a multi-attribute-based link path calculation method with the objective of reducing link latency and improving packet delivery rate. The majority of the aforementioned fault-tolerant methods do not utilize predictive information and thus cannot proactively handle faults in advance. In other domains of distributed computing, leveraging predictive information for pre-processing has demonstrated significant efficacy. In [23], author propose a distributed algorithm based on federated learning for file popularity prediction, incorporating proactive tolerance towards feedback latency. For cloud computing-based ATS, seamless recovery is essential to maintain robust service, and it requires synchronization with the active service. Leveraging predictive information enables effective pre-processing of faults, thereby achieving seamless fault recovery.

To tackle this challenge, we propose a reinforcement-based proactive reliability-aware failure recovery (PRFR) approach for the cloud-based ATS system. This method establishes a service state model based on the severity of events, and proactively implements fault recovery procedures based on the state information to achieve active fault recovery of the service. Simultaneously, the freshness of the state information is evaluated using the AoI metric to ensure the reliability and effectiveness of service management.

The main contributions of this paper are summarized as follows:

- In order to effectively tackle the ever-evolving characteristics exhibited by cloud-based ATS networks, we put forth a pioneering PRFR framework for ATS services, encompassing a triad of sequential steps for failure recovery. By formulating PRFR as an optimization problem and penalizing misbehavior, we aim to improve service reliability.
- We employ a hybrid neural network agent to proficiently address the PRFR framework and tailor it to suit our model. Additionally, we propose Age of Information (AoI) [24] to ensure information freshness and strike a balance between event occurrence and schedule time.
- The performance of the PRFR model for dynamic ATS service failure recovery is evaluated by comparing it with baseline methods for failure recovery.

The remainder of the paper is structured as follows. The second section present the architecture of the urban

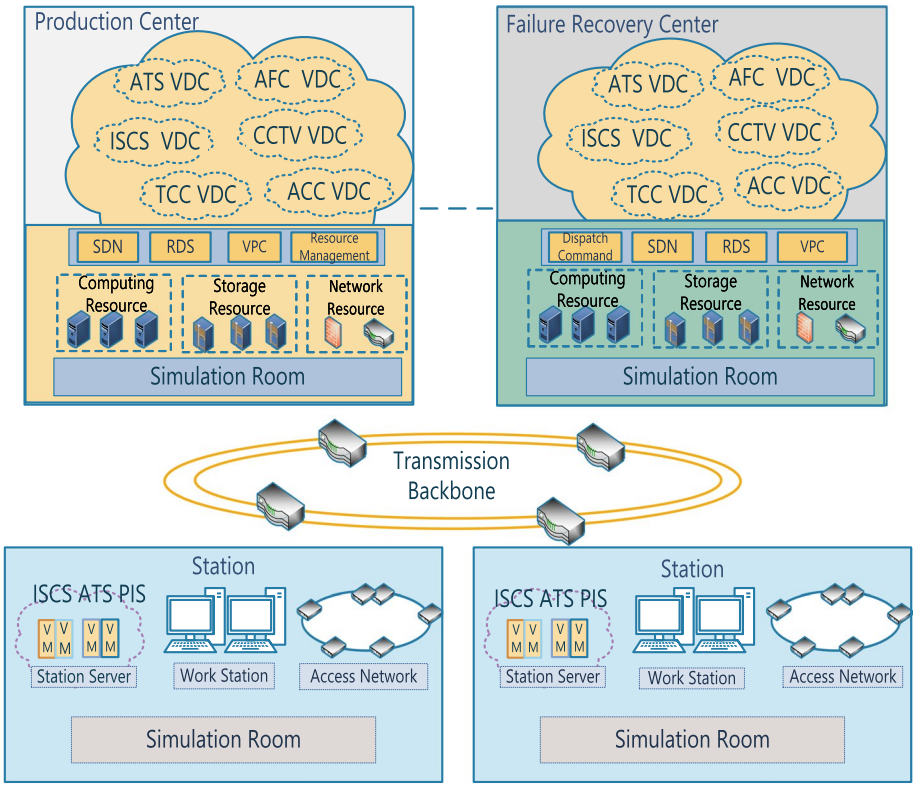

Fig. 1 The architecture of urban rail cloud

rail transit cloud platform. The third section proposed a proactive reliability-aware failure recovery procedure for cloud computing based ATS. The fourth section, we describe the system model and introduce the objective function. Section five, we present the DRL model and optimization policy. Section six, we describe the details of simulation setup and discuss the results. Finally, Section seven concludes this paper.

The cloud computing based ATS system

Urban rail transit cloud platform

The architecture of urban rail transit cloud

The integration of scattered resources through cloud computing allows the cloud platform to pool resources and enable upper-level businesses to obtain computing, storage, and other resources on demand, resulting in improved resource utilization. In the case of urban rail transit cloud, distributed cloud data centers can be deployed through a cloud management platform. The urban rail transit cloud platform typically adopts a segmented structure of data center platform-station nodes [25], which facilitates the operation, supervision, and management of the entire line.

The data center platform includes production and disaster recovery centers, located respectively in the control center and depot. Station nodes are set up at stations along the railway, and data is transmitted from the center cloud platform through the backbone ring network to the station nodes. To ensure safety, the station switches to backup mode in case of data center cloud failure. Figure 1 illustrates the architecture of the urban rail transit cloud platform.[1]

Deployment of cloud business

The architecture of the urban rail transit cloud platform is complex, as it involves the deployment of software for multiple subway lines on a uniform cloud platform. To accommodate businesses with different features, it is common practice to divide the cloud platform into separate virtual data centers (VDCs), with each VDC consisting of a private cloud for each railroad. In case of insufficient business capacity, any part of the VDC can be expanded or migrated to ensure the safe running of businesses, such as by adding CPU and storage resources.

[1] ISCS:Integrated Supervision Control System, AFC: Auto Fare Collection, CCTV:Close Circuit Television, ACC: AFC Clearing Center, TCC:Traffic Control Center

ATS in urban rail transit cloud

ATS

ATS (Automatic Train Supervision) is a critical component of the urban rail transit system, which consists of two main parts: center ATS and station ATS. The center ATS includes the control center and the disaster recovery center. *The control center responsibility for the trains' routine operations, in the event of any malfunction occurring at the control center.* On the other hand, the station ATS plays a fundamental role in automatic supervision, monitoring the status of nearby signal equipment and trains, and enabling the ATS center to dispatch the entire railroad system efficiently.

The main function of ATS includes:

- Centralized supervision. *Centralized supervision ensures the real-time depiction of railway signals and wayside equipment, while also facilitating the centralized monitoring of the interlock system and control mode at each station.*
- Timetable management. Offline editing of the basic operation diagram is available, along with automatic validity checking for created diagrams. Furthermore, the system generates an up-to-date running map based on the train's current position. By comparing this data with the planned running map, it generates the latest information and alerts as needed.
- Vehicle identification and tracking. Identifying vehicles according to schedule, ATO/ATP, etc., and monitoring section status to determine train position.
- Train and route control. Train operations are controlled by commands given to dispatchers. Provide automatic approach locking, and monitor status of signal, turnout, etc.

ATS in urban rail transit cloud

In the urban rail transit cloud, ATS still adopts the center-station architecture, mapping the business of traditional ATS to the cloud platform. Figure 2 depicts the schematic of ATS cloud deployment[2]

The cloud-based ATS system disentangles conventional ATS services and subdivides them into seven distinct microservices: universal services, application services, control services, planning services, command services, interface services, and storage services. These microservices are small, independent, and well-distributed [26]. *Information forwarding services (information centers) play a vital role in facilitating the seamless exchange of information between the microservices. In instances where there is a surge in demand for information transmission, load balancing techniques and other methodologies are employed to maintain consistent and reliable transmission. Within cloud-based ATS systems, the utilization of computing resources is optimized with greater efficiency owing to the loosely coupled nature of the system. Consequently, developers allocate computing resources solely for supplementary components when required, ensuring an optimal allocation of resources.* Additionally, technology types are no longer restricted, and different types of microservices can be organized and developed based on functional requirements. Fine-grained extensions are also possible based on actual business requirements, allowing for individual microservices to be built and maintained relatively easily. This provides full control over the ATS application business itself.

[2] CI means Computer Based Interlocking, DTI means departure time indicator.

Recovery procedure of ATS cloud platform

The current failure recovery method can be categorized into reactive and proactive approaches, each involving three main steps: launching a backup microservice, flow reconfiguration, and state synchronization. To launch a backup microservice, microservice containers are deployed for instances of failure. Flow reconfiguration requires calculating the routing path in the controller and implementing new forwarding rules. To activate backup microservices as active microservices, the ATS service gateway must be reconfigured. State synchronization involves migrating the state of failed microservices to the backup containers to support normal service. Reactive failure recovery is executed after the microservice fails, resulting in a long service recovery time due to the delay involved in the recovery procedure.

Compared to the reactive method, the proactive method reduces recovery time by predicting failures in advance. When an active microservice fails, the backup microservice, which has been pre-launched, is switched to active, and the flow is reconfigured to provide uninterrupted service. This approach allows for the complete or partial avoidance of delays in flow reconfiguration and launching microservices. The proactive method performs recovery processes earlier [13, 15], reducing the failure recovery time to the state synchronization time. The recovery procedure is shown in Fig. 3. The service link typically consists of Micser1, Micser2, and Micser3. In the event of imminent failure of Micser1 and Micser3, their backup services are launched, and the flow is reconfigured. After a service failure, state synchronization is performed.

The proposed scheme allows for the deployment and deletion of redundant backup microservices on each hardware node. Each backup microservices requires a resource allocation of h, denoted by $\varphi^{h}_{(o,s)}$ for o-th

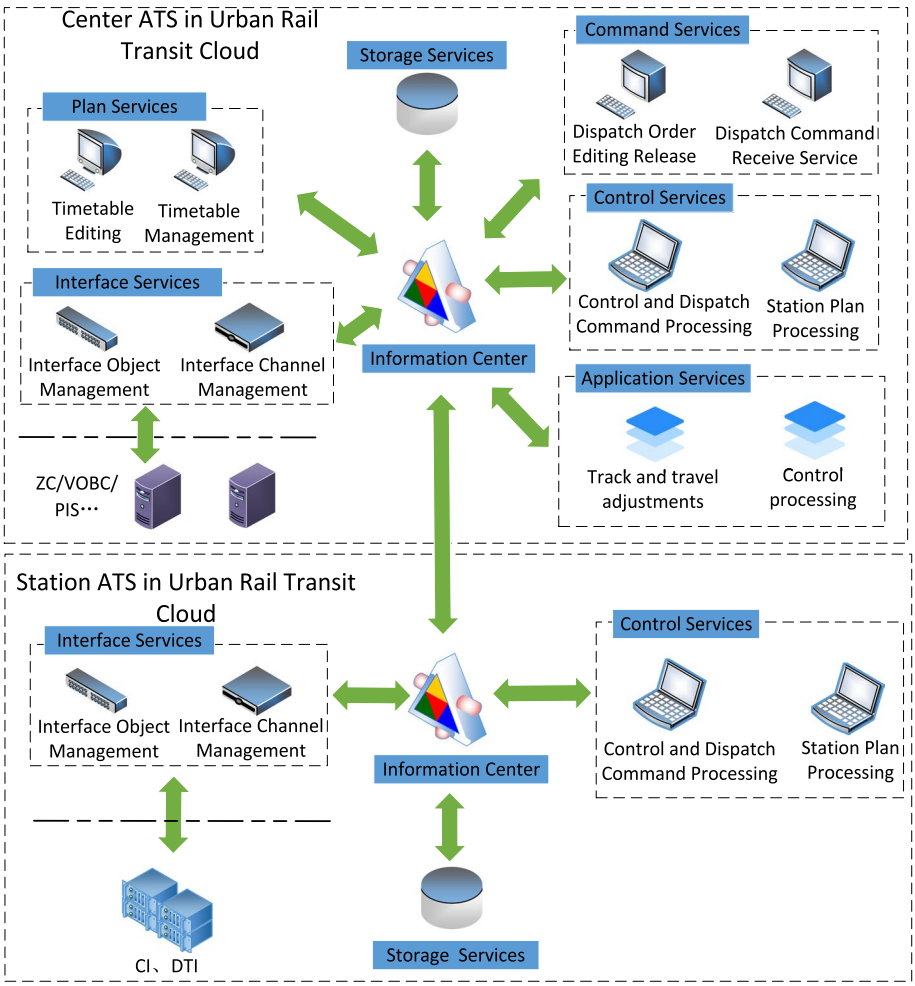

Fig. 2 The service architecture of ATS on the cloud platform

microservice in ATS service s. To ensure seamless failure recovery, the state of an active microservice is transferred to its backup. It is assumed that the rate of state updating of a microservice is linearly proportional to its packet rate ϑ_s. Each active microservice establishes a logical synchronization link with its backup microservices. To maintain these logical synchronization links $b_o^s(t)$ during the backup procedure, each link should occupy a small amount of predefined bandwidth $\varphi_{(o,s)}^{BW}$ for non-critical microservices. The value of $\varphi_{(o,s)}^{BW}$ may differ depending on the microservice's type and state freshness rate.

Assuming that the state packet size of a microservice can be observed by the orchestrator, denoted as $X_o^s(t)$, the packet needs to be transferred in time slot t to maintain synchronization between the active and backup microservices. This transfer leads to a synchronization delay, denoted as $D_o^s(t)$, between the two services.

$$D_o^s(t) = \frac{X_o^s(t)}{b_o^s(t)} \tag{1}$$

This delay is typically negligible when microservices are working correctly. However, in the event of a failure, it is crucial that the delay be shorter than the maximum tolerable interruption time μ_d, in order to avoid violating Service Level Agreement (SLA) requirements [27].

System models and problem formulation

System models

ATS network model

We model the entire ATS network as a uni-directed graph $\mathcal{G} = (\mathcal{G}^M, \mathcal{G}^L)$, where $\mathcal{G}^M$ and $\mathcal{G}^L$ represent the sets of all the network node and links. Furthermore, two separate physical nodes and their connections are indicated by $m, n \in \mathcal{G}^M$, $l_{mn} \in \mathcal{G}^L$. In the physical network, $\mathcal{G}^M$

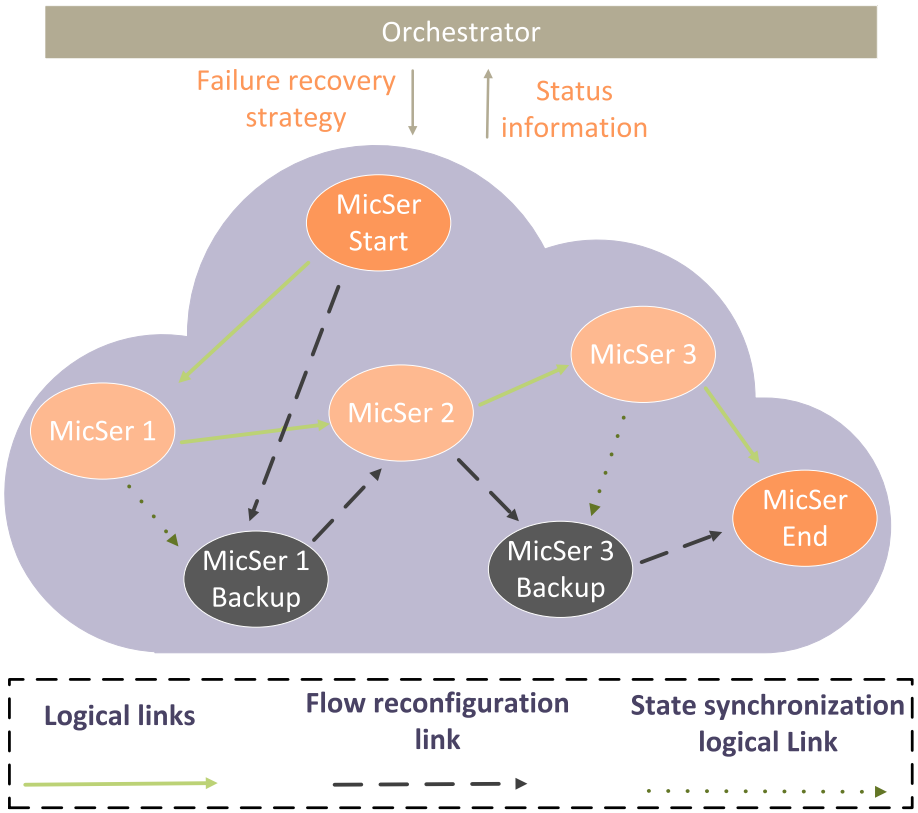

Fig. 3 An example of the proposed recovery procedure

Table 1 The main parameters table

Parameter	Definition
$M/\mathcal{G}^M/m$	Number/Node set/Physical node
$L/\mathcal{G}^L/l_{mn}$	Number of physical link/Physical link set/Physical link
$H/\mathcal{H}/h$	Number of Resources/Resource type set/Resource type
A_m^h / A_{mn}^{BW}	Maximum customizable resources h for node m / Maximum customizable bandwidth for link mn
$Q_m^h(t)$ / $Q_{mn}^{BW}(t)$	Utilization ratio of resource h in node m / bandwidth in physical link mn at time slot t
t/σ	Index/duration of time slot
$S/\mathcal{S}/s$	Traditional ATS service on cloud computing
$\mathcal{O}_s/O_s$	The set/number of sequenced microservice for ATS service s
V_o^s	The o-th microservice in service s
μ_s/ϑ_s	Maximum tolerable interruption time / traffic rate of traditional ATS service s
$\varphi_{(o,s)}^h$ / M_o^s	The number of resource type h needed / cost of a backup microservice V_o^s
$\chi_o^s(t)$	The influence ratio of the backup placement to V_o^s
$X_o^s(t)/b_o^s(t)/D_o^s(t)$	The packet size/bandwidth/delay of synchronization link of V_o^s
$k_{(o,s)}^m(t)/\check{k}_{(o,s)}^m(t)$	The backup placement of V_o^s / The placement of V_o^s
$p_o^s(t)$	If V_o^s is supported by backup
$\alpha_o^s(t)$	The recovery procedure on V_o^s

consists of the application servers, database servers, and transmission equipment. These network nodes provide processing resources for different services. In Table 1, the main parameters are listed.

In this paper, we consider a time-slotted system with positive numbers indexed by $t \in \mathcal{N}$, $\mathcal{N}$ is set of natural number. Let $\mathcal{H}$ represent the type of resource set that a physical node can provide. Each type of resource is indicated by h, $h \in \mathcal{H}$. Example of resource included: CPU, network, and storage. The maximum capacity of resource h provided by node m denoted as A_m^h, and A_{mn}^{BW} indicates the maximum bandwidth capacity of physical link l_{mn}. Considering that the available resources of a physical node can change as a result of resource releases and occupants, the current ratios of resource type h in node m indicated by $Q_m^h(t) \in [0,1]$ at each time slot t. Meanwhile, available bandwidth capacity in link l_{mn} denote as $Q_{mn}^{BW}(t) \in [0,1]$ at each time slot t.

ATS service model

As depicted in Fig. 2, the ATS services in the urban rail cloud are divided into a series of microservices that need to be combined in a specific order to provide traditional ATS services, such as timetable management. We use the set $\mathcal{S} = 1, ..., S$ to denote the set of traditional ATS services on cloud computing, indexed by s. Each sequenced ATS microservice and its corresponding SLA requirement are represented by a tuple as follows:

$$Service_s = (\mathcal{O}_s, \mu_s, \vartheta_s), \forall s \in \mathcal{S} \tag{2}$$

where $\mathcal{O}_s = \{1, ..., o_s, ..., O_s\}$ indicate the set of sequenced microservices and $\mu_s \in \mathcal{N}$ denote the maximum service interruption time, ϑ_s indicate the traffic traversing of ATS service s. And we define the set $\mathcal{V}$ consisting of all the microservices in ATS. Besides, we use $V_o^s \in \mathcal{V}$ to denote the o-th microservice in traditional ATS service s on cloud computing.

The reliability of traditional ATS services on cloud computing can be quantified as the likelihood of the microservices being executed. Failures will lead to the ATS service being degraded and the microservices being down. Our goal is to achieve proactive reliability-aware failure recovery in ATS to improve reliability. Next, we present our proactive reliability-aware failure recovery scheme for cloud computing based ATS.

ATS microservice state and state transition model

In this paper, we plan to make DRL agents capable of handling failures by utilizing state information (Table 2).

- State Model: Based on ITU standard X.733 [28], we have defined three states based on the severity of the service, namely ordinary, alert, and critical. The severity level identifies the condition of the microservice, such as CPU cycles exceeded and bandwidth reduced. When there is a change in state, the microservice send state message to orchestrator for failure management. In addition, it is worth mentioning that services of different states exhibit varying scheduling intervals to update the orchestrator's global information. The definitions of the three states are as follows: In the ordinary state, events can be ignored with no impact, and the microservice works normally. In the alert state, the microservice is degraded by software and physical events, and maintenance efforts must be made to prevent a more serious situation from arising. The critical state occurs when the severity of events reaches a serious level, implying that the failure of the microservice is unavoidable, and immediate action is required.
- Microservice State Transition Model: As shown in Fig. 4, at time slot t, if the microservice's state is ordinary, it will continue in the same state with probability $\mathcal{P}_{oo}$, or change to alert state with probability $\mathcal{P}_{oa} = 1 - \mathcal{P}_{oo}$. For possible fault and error correction, we assume the microservice will remain in an alert state for at least $\mathcal{F}_v$ time slots[3]. Furthermore, whenever the microservice remains in alert state for more than $\mathcal{F}_v$, it continues to stay alert with $\mathcal{P}_{aa}$, turns ordinary with $\mathcal{P}_{ao}$, or enters critical with $\mathcal{P}_{ac}$. We consider that the longer the microservice remains in alert state, the more likely it is to change to critical state as a result of continuous service degradation. Assume $\mathcal{P}_{ao}$ will increase by $\mathcal{P}_{ao}\times$ (step number in alet - $\mathcal{F}_v$) ≤ 1. Finally, if microservice stay in critical it will keep on until the recovery procedure is completed, and microservice turns to ordinary to provide service.

Table 2 The state transition model

State	Ordinary	Alert	Critical
Ordinary	$\mathcal{P}_{oo}$	$\mathcal{P}_{oa}$	0
Alert	$\mathcal{P}_{ao}$	$\mathcal{P}_{aa}$	$\mathcal{P}_{ac}$ grows overtime
Critical	recovery	0	No recovery

Orchestrator and AoI model

To support decision-making, it's crucial that the orchestrator has all the necessary information. When the microservice moves to the next state or reaches its scheduled time, the orchestrator receives its state packet. To ensure the

[3] The intention is to allow nodes a plausible duration for automatic recovery.

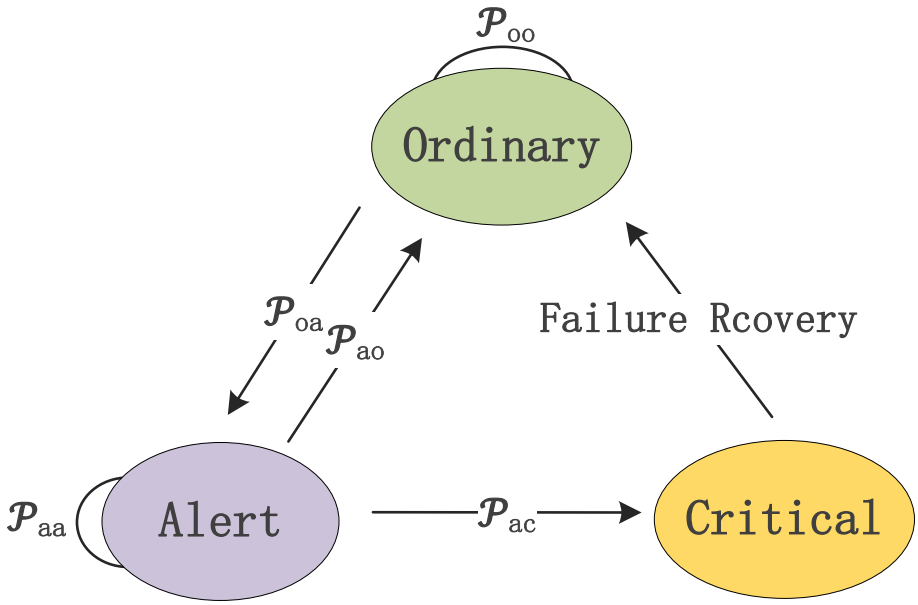

Fig. 4 State Transition Model

robustness of the orchestrator, we consider information freshness to manage microservices. Different freshness criteria are applied to different microservices, allowing the orchestrator to allocate more resources to handle microservices that occur critical events. This will enhance resource management capabilities. As the relevant information sent to the orchestrator must be updated, we use the Age of Information (AoI) metric to quantify its freshness.

Age of information (AoI) refers to the period of time between the time when information is received and the time when it was most recently generated. AoI information for microservice v is indicated by $\phi^v(t)$ at time t. During the generation of information and its transmission to the orchestrator, we assume that the network is devoid of delay [29]. AoI increases with time slots when state information is received. σ is a measure of the length of each time slot. Define AoI metric as follow:

$$\phi^v(t) = \begin{cases} \sigma & \text{beginning with time slot t} \\ \phi^v(t-1) + \sigma & \text{otherwise} \end{cases}, \tag{3}$$

During each time slot of microservices, we impose an Age of Information (AoI) constraint in which the AoI must not exceed a predetermined threshold as:

$$\phi^v(t) \leq \delta_v^s(t) \tag{4}$$

The value of $\delta_v^s(t)$ is specific to microservice v and state s at time slot t. For instance, to ensure data freshness when the microservice is in alert state, the constraint $\delta_v^s(t) \leq \mathcal{F}_v \times \sigma$ must be fulfilled. Similarly, when the microservice is in critical state, the constraint $\delta_v^s(t) \leq \sigma$ must be satisfied. The orchestrator changes the scheduling time to ensure network information freshness. By doing so, the resources used for ordinary services could be used for other more urgent services, which could result in better utilization of resources.

Problem formulation

ATS network constraints

For optimal resource utilization, all available and allocated resources in a physical node should not exceed what is currently available. Define binary variable set $\mathcal{K} = k_{(o,s)}^m(t)$ and $\mathcal{K}_{BW} = k_{(o,s)}^{mn}(t)$ for time slot t, with $k_{(o,s)}^m(t) = 1$ if the backup is placed in the node m, with the $k_{(o,s)}^{mn}(t) = 1$ if logical synchronization has been established in the physical link l_{mn}, and $k_{(o,s)}^m(t) = 0$, $k_{(o,s)}^{mn}(t) = 0$ otherwise. Here is the constrain:

$$\begin{aligned} \sum_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}}(k_{(o,s)}^m(t) - k_{(o,s)}^m(t-1)) \cdot \varphi_{(o,s)}^h \\ \leq A_m^h \cdot Q_m^h(t), \forall h \in \mathcal{H}, \forall m \in \mathcal{G}^M \end{aligned} \tag{5}$$

$$\begin{aligned} \sum_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}}(k_{(o,s)}^{mn}(t) - k_{(o,s)}^{mn}(t-1)) \cdot \varphi_{(o,s)}^{BW} \\ \leq A_m^{BW} \cdot Q_m^{BW}(t), \forall m \in \mathcal{G}^M \end{aligned} \tag{6}$$

In the first part of (5)-(6), it describes the resource of backup microservice place or release, the other part means the available resource in time slot t.

Define the $p_o^s(t)$, equals 1 if backup of V_o^s has been placed and performed recovery step 1,2.

$$p_o^s(t) = \begin{cases} 1, & \sum\limits_{m \in \mathcal{G}^M} k_{(o,s)}^m(t-1) \geq 0 \\ 0, & \sum\limits_{m \in \mathcal{G}^M} k_{(o,s)}^m(t-1) < 0 \end{cases} \tag{7}$$

In our view, active and backup microservices should never be deployed on the same physical node. This will result in meaningless backups in the event of a physical failure. To ensure the safety of this situation, we take the following constraints:

$$\sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} k_{(o,s)}^m(t) \cdot \check{k}_{(o,s)}^m(t) = 0, \forall m \in \mathcal{G}^M, \tag{8}$$

where $\check{k}_{(o,s)}^m(t)$ equals 0 if V_o^s is not placed in node m in time slot t. And in order to realize resource-efficiency, each microservice can only have one backup, expressed by:

$$\sum_{m \in \mathcal{G}^M} k_{(o,s)}^m(t) \leq 1, \forall o \in \mathcal{O}, \forall s \in \mathcal{S}, \tag{9}$$

We constrain the bandwidth of the synchronization link to ensure it does not exceed the delay threshold for critical microservices as follows:

$$0 < D_o^s(t) \leq w_o^s(t) \cdot \mu_s + \frac{1}{\tau} - w_o^s(t) \cdot \frac{1}{\tau}, \tag{10}$$

The variable $w_o^s(t)$ indicates whether V_o^s is in a critical state at time slot t. To ensure that the constraint operates properly, a small value τ is introduced.

Objective function

Considering the entire failure recovery process of the ATS network, the objective function is divided into three parts. Firstly, to prevent the placement of backup microservices after a failure occurs, we make this part as follows:

$$\xi_{SLA} = \Gamma \cdot \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} w_o^s(t) - w_o^s(t) \cdot p_o^s(t), \tag{11}$$

and Γ represents the cost associated with service interruptions and unbacked up microservices in critical states.

Backup microservices require a certain amount of resources, which are allocated by the ATS network. Assume that each backup microservice requires a distinct resource cost, denoted by M_o^s. This leads us to formulate the cost for placing a backup microservice as follows:

$$\xi_{BC} = \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} \chi_o^s(t) p_o^s(t) M_o^s, \tag{12}$$

The value of $\chi_o^s(t)$ represents the cost of backup for overutilization. The value depends on the state of the microservice. Ordinary microservices perform backup actions that do not contribute to reliability but waste more resources instead. So the value of $\chi_o^s(t)$ is considered to be high for an ordinary state. Furthermore, backing up for alert microservice is much more critical than ordinary microservice, so the value of $\chi_o^s(t)$ in alert will be lower than ordinary.

To ensure failure recovery can be completed, we define $\alpha_o^s(t)$ equals 1 if the recovery has been completed, and 0 otherwise. Whenever the orchestrator misjudges a critical microservice, a penalty Ω is imposed on the network. The third part is defined as:

$$\xi_{FR} = \Omega \cdot \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} w_o^s(t) \oplus \alpha_o^s(t), \tag{13}$$

Above all, the objective function formulate as:

$$\min_{\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{K}_{BW}} \Lambda_1 \xi_{SLA} + \Lambda_2 \xi_{BC} + \Lambda_3 \xi_{FR} \quad \textit{Subject to } (5-10) \tag{14}$$

where $\mathcal{P} = [p_o^s(t)]$, $\mathcal{A} = [\alpha_o^s(t)]$, $\mathcal{K} = [k_{(o,s)}^m(t)]$, $\mathcal{K}_{BW} = [k_{(o,s)}^{mn}(t)]$.

Deep reinforcement learning based proactive failure recovery optimization

Deep reinforcement learning framework and PRFR model

Proactive reliability-aware failure recovery (PRFR) is a complex decision-making problem that involves nonlinear constraints and integer variables in the decision variables, making it challenging to solve. However, recent advancements in reinforcement learning have enabled autonomous problem-solving without relying on human knowledge [30, 31]. In particular, deep reinforcement learning can handle high-dimensional state-action spaces by leveraging the feature extraction abilities of deep learning [32]. Therefore, we propose using DRL solutions to improve ATS failure recovery on cloud platforms.

In this paper, we choose model-free DRL. Agents explore space randomly, without prior knowledge of the environment. Policy-based reinforcement learning determines which action to take to maximize the reward function, it fine-tunes a vector of parameters noted as θ for policy π to select the appropriate action. The policy function denoted as $\pi(\alpha|s,\theta)$, represents the likelihood of selecting action α under state s and model parameters θ. After the reward feedback to agent, optimizing policy $\pi(\alpha|s,\theta)$ through gradient to fine-tune θ. Network-based ATS environments are designed to allow agents to learn a better policy, minimize the objective function, and ensure service reliability. Figure 5 illustrates the framework for deep reinforcement learning. Based on the state s, the agent performs the action α. The environment rewards the agent with r and changes the state s' accordingly. These experiences are stored as tuple (s, α, r, s') to

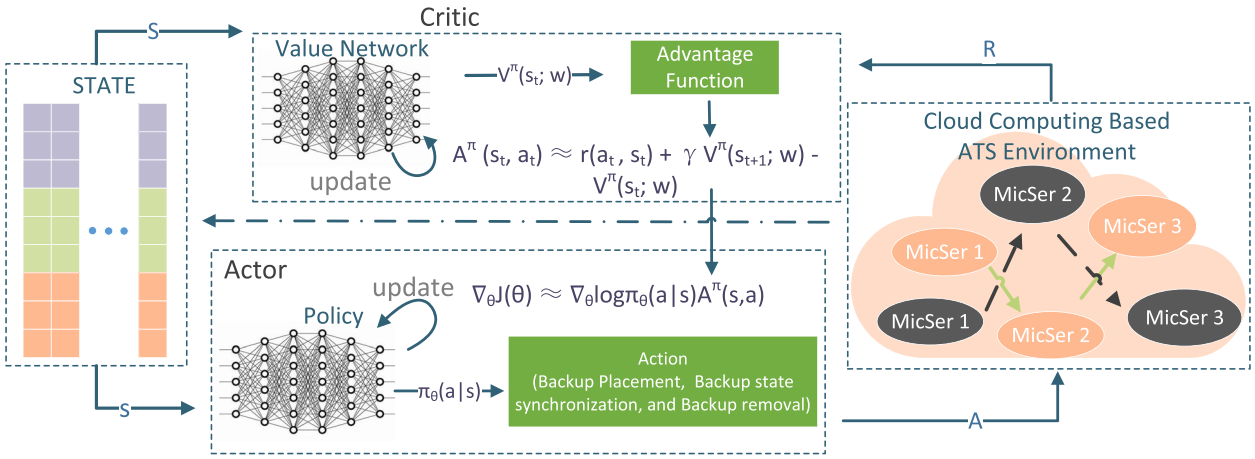

Fig. 5 The framework of DRL for ATS in cloud

use for training the policy. Here are the definitions of state, action, and reward:

State

Define three state according to state model of ATS on the cloud platform. The set of state types is defined by $S_v(t) = [1, 2, 3]$. During an ordinary state, $S_v(t) = 1$, during an alert state, $S_v(t) = 2$. and during a critical state, $S_v(t) = 3$.

Action

The agent can perform three actions: backup placement, backup state synchronization, and backup removal. Backup placement includes the first and second steps of failure recovery, while backup state synchronization indicates the final step. As soon as the recovery step is complete, the backup removal action is executed to release redundant resources. To achieve proactive reliability-aware failure recovery, the backup placement action should be executed when the microservice state is about to turn critical, and the backup state synchronization action should be executed when the microservice is in the critical state. In contrast, reactive reliability-aware failure recovery (RRFR) executes all steps after the critical state is observed.

Reward

To minimize the objective function, the agent must optimize the policies to take valuable action at each state. At time t, the agent selects an action α from a probability distribution $\pi(\alpha|s, \theta)$, where s is the current state and θ are the parameters of the agent's policy. The environment then generates a reward $R_{[all]}$, which is returned to the agent. The agent should maximize reward during each episode. The first part of the reward $R_1(t)$ defines as:

$$R_1(t) = -\Lambda_1 \xi_{SLA} - \Lambda_2 \xi_{BC} - \Lambda_3 \xi_{FR}, \quad \Lambda_1, \Lambda_2, \Lambda_3 \geq 0, \tag{15}$$

Positive rewards are defined for the agent to encourage it to take the right action. These include: executing backup removal in the ordinary microservice, rewarded as ξ_{BR}; executing backup placement before the critical state manifests, rewarded as ξ_{BP}; synchronizing state for critical microservices, rewarded as ξ_{BSS}; and successfully completing PRFR on a microservice that previously failed, rewarded as ξ_{PRFR}. Accordingly, $R_2(t)$ is defined as:

$$R_2(t) = \xi_{BR} + \xi_{BP} + \xi_{BSS} + \xi_{PRFR}, \tag{16}$$

The reward $R_{all}(t)$ defined as:

$$R_{all}(t) = R_1(t) + R_2(t). \tag{17}$$

Reinforcement learning based policy optimization for DRL model

In model-free reinforcement learning, two primary optimization methods exist: value-based and policy-based. However, value-based methods are challenging to apply to high-dimensional or continuous environments, and convergence is challenging during training. In contrast, policy-based methods allow agents to handle high-dimensional continuous environments and learn stochastic policies, thereby enhancing their exploratory abilities. Policy Gradient [33], as a fundamental algorithm of policy-based optimization, employs gradient ascent to optimize the policy function value and maximize the cumulative reward. The objective function is shown below:

$$\nabla J_\theta = E_\pi[q(S_t, A_t) \nabla log \pi_\theta(A_t|S_t)] \tag{18}$$

Policy Gradient uses iterative updates, which can be inefficient, and the sampling of a large number of trajectories

can lead to high variance. To address these issues, the Actor-Critic (AC) [34] algorithm has been developed. The AC algorithm combines value-based and policy gradient methods, and it uses two networks: the Actor network for selecting actions and the Critic network for evaluating actions. The update of Actor network is given below, α_θ is learning rate:

$$\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta ln\pi_\theta(a|s) \tag{19}$$

The objective function of Critic network is shown below, α_w is learning rate:

$$MSVE(w) = (r_t + \gamma Q_w(s',a';w) - Q_w(s,a;w))^2 \tag{20}$$

However, AC still suffers from the problem of high variance. To address this issue, the Advanced Actor-Critic (A2C) [35] method was introduced, which use the advantage function to replace the Critic network's estimate of Q values $Q_w(s,a)$. The advantage function represents the superiority $A(s, a)$ of each action value $Q(s_t, a_t)$ with respect to the mean value $V(s_t)$. This approach has shown improved efficiency and reduced variance compared to the AC algorithm. The Actor network in A2C is updated in the following way:

$$\begin{aligned} A_w(s_t,a_t) &= (Q_w(s_t,a_t) - V_w(S_t)) \\ \theta &\leftarrow \theta + \alpha_\theta A_w(s,a)\nabla_\theta ln\pi_\theta(a|s) \end{aligned} \tag{21}$$

and the update method of critic in A2C network remains the same as in AC.

1: Initial the interruption cost Γ, the maximum interruption time μ_s, the traffic traversing ϑ_s, initial policy parameters θ_0, initial value function parameters w_0, initial backup cost for overutilization $\chi_o^s(t)$, the misjudged penalty Ω, the backup resource cost M_o^s.
2: Initialize state s from environment
3: **for** $k = 0, 1, 2, ...do$ **do**
4: Sample action a according to policy $\pi(a|s;\theta)$
5: Taking action a, observe state s', reward r according to Eq. 17
6: Update θ by Eq. 21, update w by minimizing the $MSVE(w)$ according to Eq. 20
7: **end for**

Algorithm 1 A2C-based failure recovery algorithm for cloud-based ATS systems

Simulation results and discussion

Environment set up

In this paper, We use Networkx, a three-party library provided by python, to build the ATS cloud physical network nodes. We simulate four traditional ATS services on the cloud, denoted as $S = 4$, where each service consists of four microservices, $O = 4$. We consider deploy the ATS microservices in separate virtual machines (VMs), with each VM providing three types of resources: compute, storage, and network. Furthermore, we utilize four physical nodes to allocate physical resources, denoted as $\mathcal{G}^M = 4$. It is assumed that the resources required for backup placement and state synchronization are randomly assigned to each VM.

Table 3 The structure of LSTM-agent

Hidden layer type	Layer parameters
Fully connected input layer	(256, 256)
LSTM layer	(80, 80)
Fully connected output layers	(128,128)

Table 4 The structure of NLSTM-agent

Hidden layer type	Layer parameters
Fully connected input layer	(256, 256, 256, 256, 256, 256, 256, 256, 256, 128, 128, 64)
Dropout layers	(0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.3)

Each VM is represented by a state model with random transition probabilities. We define any alert VM that remains in an alert state for more than two time steps as $\mathcal{F}_v = 2$. For the alert state, we set $\delta_v^s(t) = 2$, for the critical state, we set $\delta_v^s(t) = 1$ to observe if the agent executes failure recovery successfully, and for the ordinary state, we set $\delta_v^s(t) \geq 3$. The state of the VM only changes if the scheduled time arrives or events occur, $\delta_v^s(t)$ indicating the scheduled time.

We conducted training for 75000 epochs, setting the learning rate at 2.8 x 10-2, while randomly initializing node transition probabilities within the range of 0 to 1. As for the first part of reward function $R_1(t)$, we assume $M_o^s = 1$, value of $\chi_o^s(t)$ set to 0, 0.2, 1 for critical, alert, and ordinary state, respectively. Furthermore, we set the value of Γ, Ω, and $\Lambda_1, \Lambda_2, \Lambda_3$ to 1. In the second part of the reward function $R_2(t)$, +1 reward is defined for the removal of backups in an ordinary state, +1 reward is defined for the placement of backups before critical state manifest, and +1 reward is defined for state synchronization during critical state manifests. The total of all rewards is $R_{all}(t)$. As for the agent, we use hybrid neural network structure and LSTM layers, described in Table 3. And the structure of NLSTM described in Table 4.

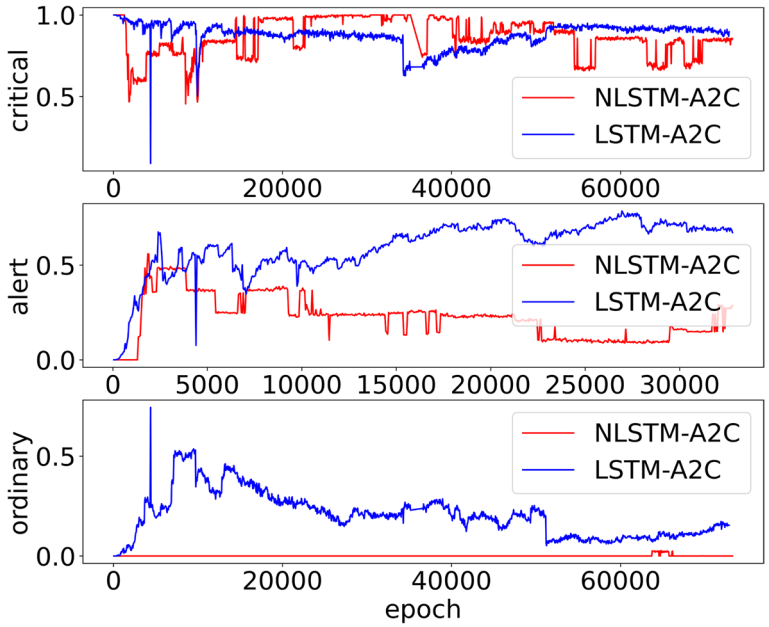

Fig. 6 Three types of accuracy comparison

Result discussion

Accuracy of different state

We have defined three state accuracy metrics to evaluate the agent's ability to monitor the VM state and take appropriate actions: A_c, A_a, and A_o, which represent critical, alert, and ordinary states, respectively:

$$\begin{cases} A_c = \frac{Na_c}{N_c} \\ A_a = \frac{Na_a}{N_a} \\ A_o = \frac{Na_o}{N_o} \end{cases} \tag{22}$$

N_c, N_a, and N_o represent the number of critical, alert, and ordinary states, while Na_c, Na_a, and Na_o denote the number of correctly taken actions in critical, alert, and ordinary states. The results are displayed in Fig. 6. The accuracy of critical states is defined as the ratio of correct actions taken in a critical state to the total number of detected critical states. We observe that LSTM-A2C and NLSTM-A2C perform similarly in critical states, with accuracy rates of 92% and 88% respectively, indicating that both models can accurately detect the state of critical virtual machines.

In terms of taking the backup placement action in the alert state, which is a measure of the accuracy of the alert state, LSTM-A2C outperforms NLSTM-A2C. When a VM enters alert state, LSTM-A2C takes backup placement action approximately 68% of the time. Additionally, LSTM-A2C performs better than NLSTM-A2C in removing backup VMs during ordinary state, leading to significant reductions in resource costs.

Failure repair rate and MTTR

In order to visualize the difference between PRFR and RRFR, we defined the failure recovery rate of PRFR and RRFR separately, as follows:

$$\begin{cases} A_{PRFR} = \frac{PRFR_r}{Nd_c} \\ A_{RRFR} = \frac{RRFR_r}{Nd_c} \end{cases} \tag{23}$$

The number of all detected critical VMs is denoted by Nd_c, while $PRFR_r$ and $RRFR_r$ represent the number of detected critical states recovered using A_{PRFR} and A_{RRFR}, respectively. The results are presented in Fig. 7, where we can observe that the LSTM-A2C method experienced

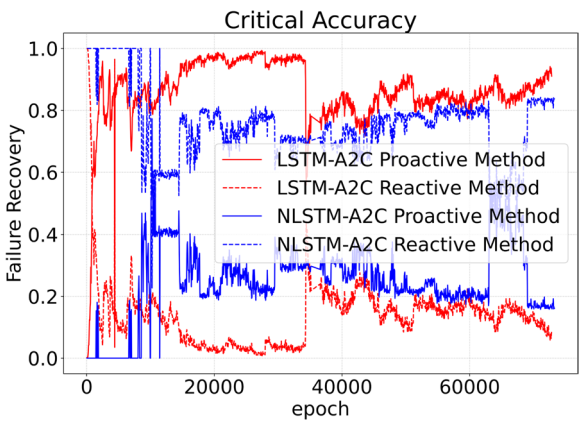

Fig. 7 Failure repair rate of PRFR and RRFR

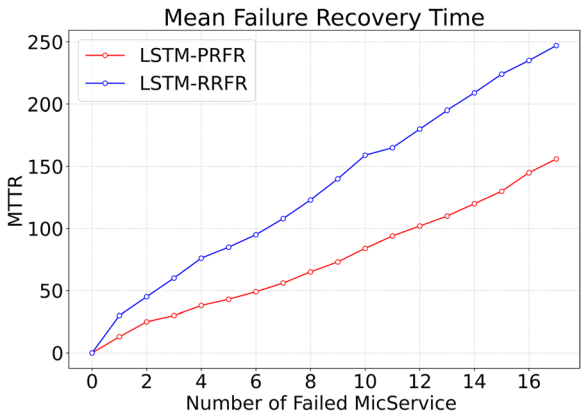

Fig. 8 MTTR of PRFR and RRFR

fluctuations around the approximate iteration count of 35,000, followed by a gradual convergence. And proactive LSTM-A2C approach achieves a higher failure recovery rate than the reactive NLSTM-A2C methods. The proactive approach enables earlier recovery actions, leading to shorter recovery times.

Meanwhile, We evaluate the performance of the model using the mean time to repair (MTTR), which measures the average time interval from the occurrence of a fault to its recovery. As shown in Fig. 8, the recovery time increases linearly with the number of failed microservices, but LSTM-PRFR exhibits a shorter recovery time than LSTM-RRFR.

Conclusion

In this paper, we investigate the architecture of the ATS on urban rail transit cloud platforms and existing reliability-based failure tolerance methods. We propose a proactive reliability-aware failure recovery method for the ATS service on the cloud platform, which takes into account both SLA infractions and resource efficiency. Secondly, we construct a state model and state transition model, considering the Age of Information to ensure the freshness of network information. We then develop a reinforcement learning model based on failure recovery steps to classify microservice states into three categories and take appropriate actions depending on their states, such as backup placement or removal. Finally, we conducted additional simulation experiments to compare our proposed model with the baseline model, and the results demonstrated that it outperformed the baseline.

Abbreviations

AoI Age of Information
ATS Automatic Train Supervision
VDC Virtual Data Center

Acknowledgements
We sincerely thank the Reviewers and the Editor for their valuable suggestions.

Authors' contributions
Li Zhu: was mainly responsible for collecting data and composing articles. Qingheng Zhuang, Hailin Jiang, Hao Liang, Xinjun Gao, Wei Wang: Mainly responsible for revising and checking articles.

Funding
No funding were used to support this study.

Availability of data and materials
No data were used to support this study.

Declarations

Ethics approval and consent to participate
This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication
The authors read and approved the final manuscript.

Competing interests
The authors declare no competing interests.

Received: 27 March 2023 Accepted: 11 August 2023
Published online: 17 October 2023

References

1. Lu K, Han B, Lu F, Wang Z (2016) Urban rail transit in china: Progress report and analysis (2008–2015). Urban Rail Transit 2:93–105
2. Liang H, Zhu L, Yu FR, Ma Z (2023) Blockchain empowered edge intelligence for TACS obstacle detection: System design and performance optimization. IEEE Trans Ind Inform 1–10. https://doi.org/10.1109/TII.2023.3257308
3. Liang H, Zhu L, Yu FR, Wang X (2023) A cross-layer defense method for blockchain empowered CBTC systems against data tampering attacks. IEEE Trans Intell Transp Syst 24(1):501–515. https://doi.org/10.1109/TITS.2022.3211020
4. Zhu L, Liang H, Wang H, Ning B, Tang T (2022) Joint security and train control design in blockchain-empowered CBTC system. IEEE Internet Things J 9(11):8119–8129. https://doi.org/10.1109/JIOT.2021.3097156
5. Zhang B, Gao S, Xia L, He J, Miao K (2010) Resource management policy for cloud testbed of china railway. In: 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), vol 4. pp V4–375–V4–379. https://doi.org/10.1109/ICCASM.2010.5619102
6. Tu H (2020) Research on the application of cloud computing technology in urban rail transit. In: 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), IEEE, pp 828–831
7. Tan X, Ai B (2011) The issues of cloud computing security in high-speed railway. In: Proceedings of 2011 International Conference on Electronic and Mechanical Engineering and Information Technology, vol 8, pp 4358–4363. https://doi.org/10.1109/EMEIT.2011.6023923
8. (2004) Ieee standard for communications-based train control (CBTC) performance and functional requirements. IEEE Std 14741-2004 (Revision of IEEE Std 14741-1999), pp 0–145. https://doi.org/10.1109/IEEESTD.2004.95746
9. Tsai WT, Zhou X, Chen Y, Bai X (2008) On testing and evaluating service-oriented software. Computer 41(8):40–46. https://doi.org/10.1109/MC.2008.304
10. Barton J, Czeck E, Segall Z, Siewiorek D (1990) Fault injection experiments using fiat. IEEE Trans Comput 39(4):575–582. https://doi.org/10.1109/12.54853
11. Zheng Z, Lyu MR (2008) A distributed replication strategy evaluation and selection framework for fault tolerant web services. In: 2008 IEEE

International Conference on Web Services, pp 145–152. https://doi.org/10.1109/ICWS.2008.42
12. Gokhale S, Trivedi K (2002) Reliability prediction and sensitivity analysis based on software architecture. In: 13th International Symposium on Software Reliability Engineering, 2002. Proceedings, pp 64–75. https://doi.org/10.1109/ISSRE.2002.1173214
13. Natalino C, Coelho F, Lacerda G, Braga A, Wosinska L, Monti P (2018) A proactive restoration strategy for optical cloud networks based on failure predictions. In: 2018 20th International Conference on Transparent Optical Networks (ICTON), pp 1–5. https://doi.org/10.1109/ICTON.2018.8473938
14. Yacoub S, Cukic B, Ammar H (1999) Scenario-based reliability analysis of component-based software. In: Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443), pp 22–31. https://doi.org/10.1109/ISSRE.1999.809307
15. Huang H, Guo S (2019) Proactive failure recovery for NFV in distributed edge computing. IEEE Commun Mag 57(5):131–137. https://doi.org/10.1109/MCOM.2019.1701366
16. Liu J, Wang S, Zhou A, Kumar SAP, Yang F, Buyya R (2018) Using proactive fault-tolerance approach to enhance cloud service reliability. IEEE Trans Cloud Comput 6(4):1191–1202. https://doi.org/10.1109/TCC.2016.2567392
17. Zhang P, Shu S, Zhou M (2018) An online fault detection model and strategies based on SVM-grid in clouds. IEEE/CAA J Autom Sin 5(2):445–456. https://doi.org/10.1109/JAS.2017.7510817
18. Zhou A, Wang S, Cheng B, Zheng Z, Yang F, Chang RN, Lyu MR, Buyya R (2017) Cloud service reliability enhancement via virtual machine placement optimization. IEEE Trans Serv Comput 10(6):902–913. https://doi.org/10.1109/TSC.2016.2519898
19. Fan J, Guan C, Zhao Y, Qiao C (2017) Availability-aware mapping of service function chains. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp 1–9. https://doi.org/10.1109/INFOCOM.2017.8057153
20. Zhang X, Wu C, Li Z, Lau FC (2017) Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp 1–9. https://doi.org/10.1109/INFOCOM.2017.8057118
21. Yang L, Li Y, Yang SX, Lu Y, Guo T, Yu K (2022) Generative adversarial learning for intelligent trust management in 6g wireless networks. IEEE Netw 36(4):134–140. https://doi.org/10.1109/MNET.003.2100672
22. Zhao L, Yin Z, Yu K, Tang X, Xu L, Guo Z, Nehra P (2022) A fuzzy logic-based intelligent multiattribute routing scheme for two-layered sdvns. IEEE Trans Netw Serv Manag 19(4):4189–4200. https://doi.org/10.1109/TNSM.2022.3202741
23. Lin N, Wang Y, Zhang E, Yu K, Zhao L, Guizani M (2023) Feedback delay-tolerant proactive caching scheme based on federated learning at the wireless edge. IEEE Netw Lett 5(1):26–30. https://doi.org/10.1109/LNET.2023.3237261
24. Kosta A, Pappas N, Angelakis V (2017) Age of Information: A New Concept, Metric, and Tool
25. Biao W,(2019) The disaster preparedness scheme under the urban rail cloud architectur. Urban Rapid Rail Transit 3:25
26. Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Casallas R, Gil S (2015) Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 2015 10th Computing Colombian Conference (10CCC), pp 583–590. https://doi.org/10.1109/ColumbianCC.2015.7333476
27. Qu K, Zhuang W, Ye Q, Shen X, Li X, Rao J (2020) Dynamic flow migration for embedded services in SDN/NFV-enabled 5g core networks. IEEE Trans Commun 68(4):2394–2408. https://doi.org/10.1109/TCOMM.2020.2968907
28. Union IT (1992) Information technology—open systems interconnection—system management: Alarm reporting function
29. Kadota I, Sinha A, Modiano E (2018) Optimizing Age of Information in Wireless Networks with Throughput Constraints. IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. Honolulu, pp 1844–1852. https://doi.org/10.1109/INFOCOM.2018.8486307
30. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A et al (2017) Mastering the game of go without human knowledge. Nature 550(7676):354–359
31. Berner C, Brockman G, Chan B, Cheung V, Dębiak P, Dennison C, Farhi D, Fischer Q, Hashme S, Hesse C, et al (2019) Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680
32. Shrestha A, Mahmood A (2019) Review of deep learning algorithms and architectures. IEEE Access 7:53040–53065
33. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: International conference on machine learning, PMLR, pp 387–395
34. Konda V, Tsitsiklis J (1999) Actor-critic algorithms. Adv Neural Inf Process Systems 12. https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.
35. Sutton R, Barto A (2018) Reinforcement learning: An introduction. MIT Press, Google Scholar, pp 329–331

Publisher's Note