Shi JC, Cai XQ, Zheng WL *et al.* Reliability and incentive of performance assessment for decentralized clouds. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(5): 1176–1199 Sept. 2022. DOI 10.1007/s11390-022-2120-y

Reliability and Incentive of Performance Assessment for Decentralized Clouds

Jiu-Chen Shi[1] (史久琛), *Student Member, CCF, IEEE*, Xiao-Qing Cai[1] (蔡晓晴), *Student Member, CCF, IEEE*
Wen-Li Zheng[1] (郑文立), *Senior Member, CCF, Member, IEEE*
Quan Chen[1] (陈　全), *Senior Member, CCF, Member, IEEE*
De-Ze Zeng[2] (曾德泽), *Senior Member, CCF, Member, IEEE*, Tatsuhiro Tsuchiya[3], *Member, IEEE*, and
Min-Yi Guo[1,*] (过敏意), *Fellow, CCF, IEEE*

[1]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*
[2]*School of Computer Science, China University of Geosciences, Wuhan 430074, China*
[3]*Graduate School of Information Science and Technology, Osaka University, Osaka 565-0871, Japan*

E-mail: {shijiuchen, cai-xq}@sjtu.edu.cn; {zheng-wl, chen-quan}@cs.sjtu.edu.cn; deze@cug.edu.cn
t-tutiya@ist.osaka-u.ac.jp; guo-my@cs.sjtu.edu.cn

Received December 28, 2021; accepted July 21, 2022.

Abstract Decentralized cloud platforms have emerged as a promising paradigm to exploit the idle computing resources across the Internet to catch up with the ever-increasing cloud computing demands. As any user or enterprise can be the cloud provider in the decentralized cloud, the performance assessment of the heterogeneous computing resources is of vital significance. However, with the consideration of the untrustworthiness of the participants and the lack of unified performance assessment metric, the performance monitoring reliability and the incentive for cloud providers to offer real and stable performance together constitute the computational performance assessment problem in the decentralized cloud. In this paper, we present a robust performance assessment solution RODE to solve this problem. RODE mainly consists of a performance monitoring mechanism and an assessment of the claimed performance (AoCP) mechanism. The performance monitoring mechanism first generates reliable and verifiable performance monitoring results for the workloads executed by untrusted cloud providers. Based on the performance monitoring results, the AoCP mechanism forms a unified performance assessment metric to incentivize cloud providers to offer performance as claimed. Via extensive experiments, we show RODE can accurately monitor the performance of cloud providers on the premise of reliability, and incentivize cloud providers to honestly present the performance information and maintain the performance stability.

Keywords decentralized cloud computing, robust performance assessment, trusted execution environment (TEE)

1 Introduction

Cloud computing, as a typical resource sharing solution, delivers various forms of computing services across the Internet, while four cloud giants (i.e., AWS, Microsoft, Alibaba, and Google) dominate 77.3% of the market share①. Moreover, it is discovered that as much as 85% of the server capacity is underutilized[1], especially for small or medium enterprises and cloud providers. Recently, lots of decentralized cloud platforms have been implemented, e.g., Akash②, BonusCloud③, Golem④, iExec⑤, and

Regular Paper
This work is supported by the National Natural Science Foundation of China under Grant Nos. 61832006 and 61872240.
*Corresponding Author

①Gartner says worldwide IaaS public cloud services market grew 37.3% in 2019. https://www.gartner.com/en/newsroom/press-releases/2020-08-10-gartner-says-worldwide-iaas-public-cloud-services-market-grew-37-point-3-percent-in-2019, May 2022.

②https://akash-web-prod.s3.amazonaws.com/uploads/2020/03/akash-position.pdf, May 2022.

③https://bonuscloud.io/whitepaper/BonusCloud%20White%20Paper%202018%20Version%201.0.pdf, May 2022.

④https://whitepaper.io/document/21/golem-whitepaper, May 2022.

⑤https://iex.ec/wp-content/uploads/pdf/iExec-WPv3.0-English.pdf, May 2022.

SONM[6], which have the potential to utilize idle computing power across the Internet, no matter personal computers or enterprise servers. With the same network architecture (i.e., peer-to-peer) and the same purpose (i.e., combining idle resources) of grid computing or peer-to-peer computing, these decentralized cloud platforms are enhanced by exploring the newly emerged Blockchain technology [2] to form the reliable supervision of computing power trading. The integration of the Blockchain technology can further guarantee the system's robustness in a decentralized and de-trust mode.

Computational performance is a critical metric to assess the quality of cloud computing [3, 4], and is even more important in decentralized cloud platforms. Since any user or enterprise can be the cloud provider, comparing the performance between heterogeneous computing resources is essential. Cloud providers can label their servers with respective performance information, e.g., the results of executing some widely-used benchmarks. Usually, a user needs to pay more for renting a cloud server with better performance, and hence expects the actual performance to be as good as claimed.

However, the cloud providers in the decentralized cloud may not always keep the performance at the claimed value in practice, mainly due to the following two reasons. Firstly, resource sharing and contention can affect the performance stability of the claimed performance [5, 6]. Secondly, if the cloud providers participating in the decentralized cloud have no requirement of trust or reputation, a malicious cloud provider can intentionally claim inveracious performance information to gain more profit [7, 8]. These two facts lead to the untrustworthiness of the claimed performance, and thus the monitoring of real performance becomes indispensable in the decentralized cloud.

To address such an untrustworthiness problem, several methods have been proposed to assess the real performance of untrustworthy cloud providers. As shown in the left part of Fig. 1, a user who does not trust the cloud providers can monitor the performance through his/her workload executions and rate the cloud providers onto the Blockchain. However, the user's results are hard to be comprehensive and may not be trusted by other users and cloud providers. Some third parties can benchmark the performance of cloud providers (e.g., SPEC rating) and then record the results onto the Blockchain, as shown in the right part of Fig. 1. This method is not transparent and requires strong trust, while the cloud providers can easily conduct malicious behaviors, e.g., tampering with the benchmarking profiles or offering better VMs for benchmarking. In the decentralized cloud, although the performance information (users' rates or benchmark results) recorded on the Blockchain is traceable and non-tampered, there is no guarantee on its quality. The performance information does not come from a reliable

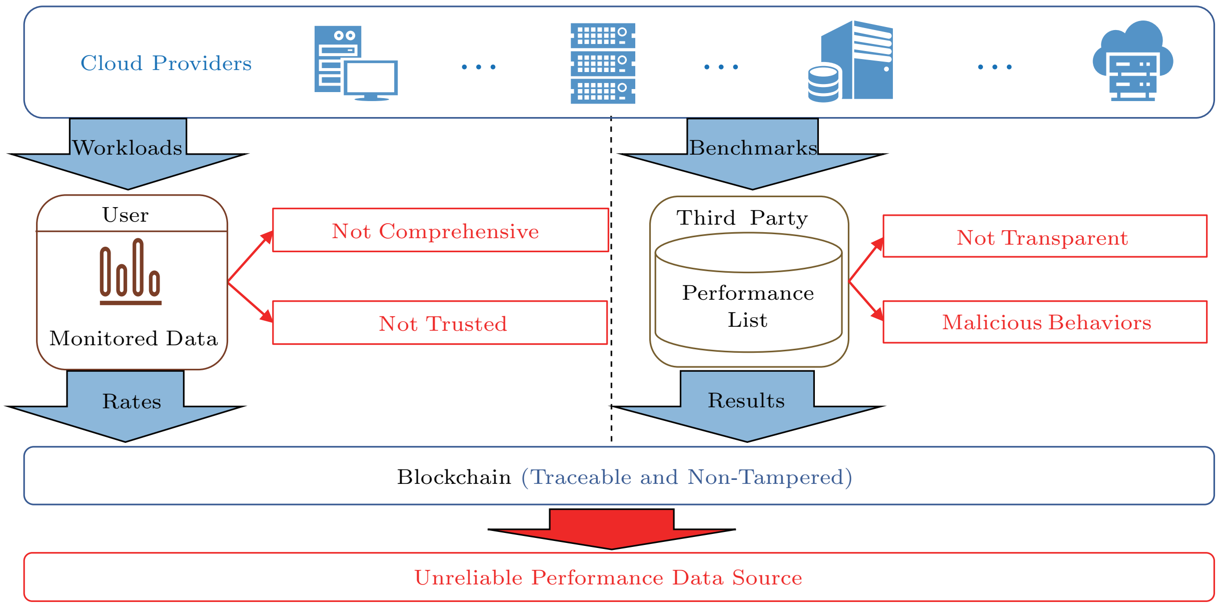


Fig. 1. Existing methods of acquiring performance information in decentralized clouds. Both of the methods have problems with reliability.

[6]https://docs.sonm.com/, May 2022.

data source, because none of the cloud providers, users, or third parties can provide reliable performance information. Therefore, acquiring reliable performance monitoring results in the decentralized cloud is still challenging.

Meanwhile, performance assessment based on performance monitoring results is also of vital significance to the incentive mechanism of the decentralized cloud. Typically, cloud providers with better performance shall get more revenue by providing higher Quality-of-Service (QoS) to the users. Existing peer-to-peer computing or decentralized computing proposes economy-based [9, 10] and reciprocity-based [11–13] incentive mechanisms, and includes various theories and algorithms, e.g., game theory [14, 15] and double auction [16]. These incentive mechanisms utilize the reputation or contribution (e.g., the historical ratio of successful interactions) of the peers as the input, to avoid the malicious behaviors of peer-to-peer systems. However, these incentive mechanisms do not consider performance assessment as an incentive factor. Moreover, existing incentive mechanisms cannot be directly applied due to the unreliable input (cloud providers' performance monitoring results) and the lack of suitable performance assessment metric for the incentive mechanisms. Since there is no robust and adaptive incentive mechanism for performance assessment, the participants in the decentralized cloud may claim inveracious performance information and have no motivation to maintain the performance stability with respect to their claimed performance. This may even raise the phenomenon that bad money drives out the good. Therefore, a unified performance assessment metric, which is generated based on reliable performance monitoring results, is needed to fairly and accurately assess the cloud providers, and shall also be taken into account in the cloud provider selection decision from users.

In this paper, we present a robust performance assessment solution RODE. We first propose a performance monitoring mechanism to generate reliable and verifiable performance results for workload executions on the untrusted cloud providers. A delicate sampling solution empowered by Intel Software Guard Extensions (SGX) technology [17, 18] is proposed to realize reliable performance monitoring in the specific trusted execution environment (TEE) [19], making the performance data source reliable. We use two common characteristics of Intel SGX existing in most TEEs on various CPU processor architectures (e.g., Intel TDX [20], AMD SEV-SNP [21], and ARM CCA [22]), to enable that our mechanism can be easily adapted to many different types of servers in the decentralized cloud. We focus on batch workloads with the Infrastructure as a Service (IaaS) mode and cloud providers' computational performance. In decentralized clouds, the claimed hardware configuration of providers may be different. Given a user's workload, since we cannot obtain the user's workload corresponding execution time (end-to-end delay) that matches the given claimed hardware resources, we cannot tell whether the resources are truly provisioned as the cloud providers claimed. Therefore, rather than the execution time, we mainly use the micro benchmarks inserted in the users' workloads as samplings, to obtain the performance monitoring results. Based on the reliable performance results, we further propose the assessment of the claimed performance (AoCP) mechanism to form a unified performance assessment metric among cloud providers, and then use this metric in the cloud provider selection decision for users. In this way, the cloud providers are incentivized to offer real and stable performance.

We evaluate the performance monitoring mechanism to show that it can sensitively detect whether the workload execution meets the claimed performance or not, with negligible overhead. We also validate its practicability under dynamic resource sharing of the cloud environment. In addition, we emulate how users choose cloud providers according to the dynamic performance assessment with AoCP, and show that RODE can incentivize cloud providers to honestly present the performance information and maintain the performance stability at runtime. Specifically, the major contributions of this paper are as follows.

• We design a performance monitoring mechanism, as a trusted method to monitor the workload execution performance on untrusted cloud providers, to generate reliable and verifiable performance results. It offloads the workloads and performance monitoring into untrusted memory and only uses SGX to hide the critical part of the performance monitoring, to avoid SGX's limitations and improve the performance monitoring effect.

• We design the AoCP mechanism, as a performance assessment method for cloud servers based on reliable performance monitoring results, to incentivize cloud providers to offer real and stable performance.

• We implement the SGX-based performance monitoring mechanism and emulate the AoCP mechanism. The experimental results show the feasibility of trusted performance monitoring and the effectiveness of AoCP.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the background and motivation. Section 4 describes the design of RODE. Section 5 is the evaluation. Section 6 provides the discussions. Section 7 concludes this work.

2 Related Work

The performance or QoS information is critical for comparing the cloud providers with heterogeneous computing power in the decentralized cloud. Most existing decentralized cloud platforms use simple benchmarking (e.g., iExec and SONM) or performance monitoring and pressure testing (e.g., BonusCloud) to monitor the performance of untrusted cloud providers [8]. However, without any reliability guarantee, a malicious cloud provider can easily report inveracious performance to others, e.g., tampering with the performance benchmarking profiles.

Some existing studies consider the cloud service selection, composition, and recommendation, and acquire or monitor the performance or QoS information in different ways. The widely-used centralized schemes [23–25] support obtaining the information from trusted third parties or the cloud providers themselves, which requires strong trust, while it is totally possible for a third party and a cloud provider to operate in collusion. Some studies present how to estimate the performance by modeling with historical data, related attributes, or context information, from the user side or the cloud provider side [26–30]. However, neither side can ensure the reliability of the data and models used for estimation, and hence neither can convince the other when their results are inconsistent. Some other studies adopt the smart contract [31] technology to monitor the response time of latency-critical services and record it on the Blockchain [32, 33]. However, for most batch workloads, more performance related data (e.g., CPU cores or memory bandwidth usage), not only the response time, are needed to accurately assess the performance, which cannot be monitored by the smart contract. Therefore, existing studies cannot verify the reliability of the performance data due to the untrusted execution of performance monitoring.

Meanwhile, the emergence of the trusted execution environment (TEE) [19] brings opportunities for reliable performance monitoring on the untrusted hosts. Intel SGX [17, 18] is one of the technologies providing a trusted execution environment [19], with which a user's application program can execute in the secured memory called enclave to ensure the correctness and integrity of execution. The code, input data, and user-supplied data can be measured and included in a CPU-signed report provided by Intel. The user can use the CPU-signed report to issue a verification request to Intel Attestation Service (IAS) for remote attestation, which can conduct authentication of the remote enclave of the remote cloud provider. The SGX's major functionalities with secured memory protection and remote attestation also exist in most of the other TEEs on various CPU processors, e.g., Intel TDX [20], AMD SEV-SNP [21], and ARM CCA [22]. With the rapid development of Intel SGX technology, OVERSEE [7] proposes an SGX-based performance monitoring mechanism. However, it requires the entire workload and the performance monitoring executing inside the SGX enclave. This brings several limitations to the workload (e.g., memory space, IO instructions, and system calls), resulting in severe monitoring errors and overhead.

Except for the performance monitoring reliability, the performance assessment is also of vital significance for incentivizing cloud providers to offer good performance. Existing incentive mechanisms of peer-to-peer or decentralized cloud platforms can be classified into two categories, i.e., economy-based and reciprocity-based. Economy-based mechanisms [9, 10] price shared entities (e.g., computing resources) according to participants' provided service performance or the supply and demand situation. On the one hand, the participant who can provide better quality of experience (QoE) is allowed to offer higher prices of shared entities to gain more profits [9]. On the other hand, some mechanisms use various game theories [14, 15] or double auction techniques [16] to control shared entities' prices and bid to achieve a balanced supply and demand. Reciprocity-based mechanisms [11–13] enable a participant to get equivalent rewards according to the participant's contribution to the system. For instance, the participant who contributes more hot data for service caching can get a higher reputation, and in return, the participant can obtain a better quality of service from the edge server [11]. The above two categories utilize the price, contribution, or reputation of participants as the input, to avoid the malicious behaviors of the peer-to-peer system. However, existing incentive mechanisms fail to take the performance assessment into consideration. Moreover, they cannot be directly applied due to the unreliable input (cloud provider's performance monitoring results) and no suitable performance assessment metric for the incentive mechanisms. Since there are no robust and adaptive incentive mechanisms for performance assessment, the cloud providers have no motivation to offer high performance for users.

Compared with existing schemes, RODE not only exploits Intel SGX to guarantee the reliability of performance data source while eliminating the poor effect caused by Intel SGX by the delicate design, but also takes advantage of the performance assessment attribute to incentivize cloud providers to offer real and stable performance, so as to form a robust performance assessment solution for the decentralized clouds.

3 Background and Motivation

In this section, we introduce the decentralized cloud platforms and the motivation behind the design of RODE.

3.1 Decentralized Cloud Platforms

The four cloud giants, i.e., AWS, Microsoft, Alibaba, and Google, take up most of the cloud market share, but their infrastructures require a huge amount of cost. Small and medium cloud providers, enterprises, or personal computers across the network have a large number of idle computing resources, but their competitiveness is insufficient, due to the lack of reputation, investment, and so on. To avoid vendor lock-in, reduce the cloud service cost, and efficiently utilize the idle resources across the network, many decentralized cloud platforms are emerging, e.g., Akash, BonusCloud, Golem, iExec, and SONM. Expect for building on the peer-to-peer network to effectively aggregate idle resources worldwide like grid computing and peer-to-peer computing, these platforms enable reliable computing power trading by utilizing the Blockchain technology to transparently provide cloud services to users.

The operation flow of a typical decentralized cloud platform is shown in Fig. 2. A cloud user first submits the workload requests to the transaction manager, which then broadcasts a corresponding transaction (steps 1 and 2). After this transaction is processed and recorded on the Blockchain, the scheduler can obtain the user's workload requirements from this transaction, e.g., the required number of CPU cores, memory capacity, or IO bandwidth (step 3). The scheduler then chooses the most appropriate cloud provider according to the selection policy, which considers three aspects: 1) resources and price the cloud providers can provide, 2) the cloud providers' performance from the performance collector, and 3) the cloud providers' reputation level from the incentive system (step 4). Thereafter, the selected cloud provider executes the workload and sends the results to the result verifier (steps 5 and 6). If the verification is failed, the workload request will be sent back to the scheduler and be re-scheduled. In this case, the reputation of the corresponding cloud provider decreases. Otherwise, the transaction manager will send transactions to pay for the cloud provider and the reputation value increases (steps 7 and 8).

The result verification realized by the result verifier plays a critical role in decentralized cloud platforms. Since the cloud providers in the decentralized cloud

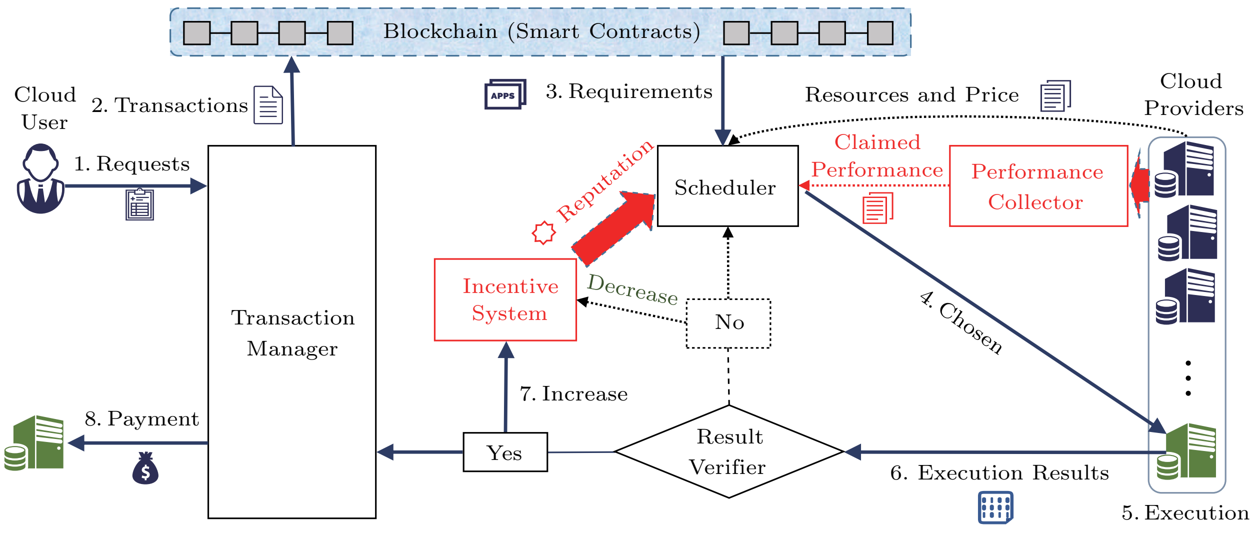


Fig.2. Simplified operation flow of a decentralized cloud. The parts in red have the problems we study, i.e., the reliability of performance monitoring (Subsection 3.2.1 and Subsection 3.2.2) and the incentives of performance assessment (Subsection 3.2.3).

platform have a low barrier to entry, the trustworthiness of them cannot be guaranteed. The execution results from the cloud providers may be inveracious. To solve this problem, Golem, iExec, and SONM use log analysis, correctness checking, and redundant computing methods. Furthermore, these platforms will take the number of correctly completed workloads as the main factor to rate the reputation of cloud providers, and take the reputation into account in the incentive system to avoid malicious behaviors. However, these methods cannot completely guarantee the correctness and also cost a huge amount of extra computing power. In addition, iExec and Golem attempt to integrate Intel SGX technology to protect the integrity of the code and guarantee the execution results[7][8].

In this paper, we focus on the performance assessment problems in decentralized cloud platforms, which exist in the performance collector and the incentive system of Fig.2.

3.2 Performance Assessment Problems

This subsection analyzes two key problems, i.e., the reliability of performance monitoring (Subsection 3.2.1 and Subsection 3.2.2) and the incentives considering performance assessment (Subsection 3.2.3), which exist in the performance collector and the incentive system of Fig.2, respectively.

3.2.1 Unreliable and Unverifiable Performance Monitoring

In the decentralized cloud, each cloud provider may have multiple types of cloud servers with different performance scores. A cloud provider can measure the performance by executing public benchmarks and converting the benchmark results into performance scores, which are then used to label the cloud server as claimed performance. The user selects the cloud server based on the claimed performance and expects it can be achieved for the workloads submitted. However, the actual performance may not always be consistent with the claimed value due to the resource sharing of the cloud servers and the cloud providers' malicious behaviors to gain more profit. For example, an enterprise server as the cloud provider may get and claim its memory performance when the server is idle, but perform its own big data processing business at the same time when executing users' workloads obtained from the decentralized cloud, which can cause the contention of memory bandwidth. Another example, a malicious cloud provider may claim its server has the CPU float calculation performance of 5 GFLOPS, but only uses lower CPU performance to perform users' workloads, which can not only achieve a high service sale price but also save server energy. These situations all lead to the untrustworthiness of the claimed performance.

To address this problem, usually a user can monitor the user workload's performance via system calls (e.g., similar to what the Linux command *top* does), which are conducted on the cloud servers. Unfortunately, an untrusted cloud provider can manipulate the system calls to forge the performance information, leading to unreliable performance monitoring results. Meanwhile, it is also difficult for the cloud providers to prove the validity of their performance results.

For the performance collector in Fig.2, some cloud platforms directly adopt the claimed performance from the cloud providers. With respect to the trustworthiness issue, some platforms suggest benchmarking methods (e.g., iExec and SONM), or using performance monitor plugins and pressure test methods (e.g., BonusCloud) to actively monitor the performance [8]. These methods can all be easily disrupted by malicious behaviors and hard to be validated, and thus lack of reliability and verifiability.

3.2.2 High Error Rate and Overhead of Existing SGX-Based Solution

Intel SGX can isolate sensitive codes and data in the protected memory, called enclave, and provide a CPU-signed report for the code running in the enclave. Hence, OVERSEE [7] proposes a state-of-the-art SGX-based performance monitoring mechanism, which addresses the mutual distrust problem in performance monitoring to guarantee the reliability and verifiability. To this end, OVERSEE is designed to execute the entire workload inside SGX's enclave, and evenly insert performance samplings with micro benchmarks into the workloads. OVERSEE calculates the number of benchmarks executed per second by looping to get SGX's trusted timestamps (with one-second granularity) in each sampling, and uses all samplings' average value to represent the workload's performance.

However, executing the entire workload inside SGX causes non-negligible impacts on the workload's performance, and brings many limitations, e.g., it is hard to

[7]https://docs.iex.ec/for-developers/confidential-computing/intel-sgx-technology, May 2022.

[8]https://blog.golemproject.net/golems-essential-guide-to-graphene/, May 2022.

support IO instructions and system calls. Moreover, we discover that OVERSEE is not accurate when there exists resource contention with other co-located workloads on the same server. We reproduce OVERSEE and test a CPU-intensive workload π calculation and a memory-intensive workload BubbleSort in six different interference scenarios, respectively. We insert float calculation benchmark into π calculation, and array copy (1D) benchmark into BubbleSort. The experiments are conducted on a PC server to be consistent with OVERSEE[7]. The hardware and software specifications, workloads and benchmarks, and interference scenarios are shown in Table 1, Table 2, and Table 3 respectively.

Fig.3 reports our experimental results on both errors and overhead. We can see that both the error rate and the overhead of OVERSEE are high. The average error rate is 24.8% and the average overhead is 91.8% for π calculation, while, the average error is 26.7% and the average overhead is 132.2% for BubbleSort. We attribute this phenomenon to the following two reasons. The first one is the inaccurate timestamps OVERSEE uses. The granularity of SGX's trusted timestamps is 1 s, and the time to obtain SGX's timestamp differs greatly under different resource contention from the co-located workloads. In our experiments, the time to obtain an SGX's timestamp spans from 13.3 ms to 23.4 ms. Such an inaccurate timestamp incurs a large error in OVERSEE. The second one is the SGX's overhead. OVERSEE executes workloads in SGX inherently, which incurs too much inevitable SGX's overhead (e.g., limitations on memory space with 128 MB) and performance sampling overhead (1 s–2 s for each sampling).

Table 1. Hardware and Software Specifications

Server Type	Specification
PC server	Intel Core i7-6700 CPU @ 3.4 GHZ, 4 hyper-threaded cores, 16 GB DDR4 2 133 MHz, Ubuntu 20.04.2 LTS, Docker version 19.03.12, tozd/sgx[9]
High-performance server	Intel Xeon Gold 5218 CPU @ 2.30 GHz, 32 hyper-threaded cores, 256 GB DDR4 2 133 MT/s, Ubuntu 18.04.5 LTS, Docker version 19.03.12

Table 2. Workloads and Benchmarks in Testbed Experiments

	Workload Name	Specification
CPU-intensive workload	π calculation	45 000 digits
	SVM train	Mnist-str4 dataset
Memory-intensive workload	BubbleSort	1 MB data
	WordCount	162 061 lines
CPU-intensive benchmark	Float calculation	1.5×10^7 interactions of calculations (plus, subtract, multiply, and divide)
Memory-intensive benchmark	Array copy (1D)	Memory copy of two one-dimensional arrays with 24 MB
	Array copy (2D)	Memory copy by columns of two two-dimensional arrays with 8 MB

Table 3. Interference Scenarios for Comparison with Baseline

Scenario	π Calculation (CPU-Intensive)	BubbleSort (Memory-Intensive)
1	2 CNN+2 RL+1 K-means	Redis (1 port)
2	3 CNN+3 RL+2 K-means	Memcached (2 ports)
3	3 CNN+3 RL+3 K-means	Redis (1 port)+ Memcached (1 port)
4	4 CNN+4 RL+3 K-means	Redis (1 port)+ Memcached (2 ports)
5	5 CNN+4 RL+4 K-means	Redis (2 ports)+ Memcached (1 port)
6	5 CNN+5 RL+5 K-means	Redis (2 ports)+ Memcached (2 ports)

Note: CNN is short for convolutional neural network training. RL is short for reinforcement learning. Redis workloads are generated by using the load generation tool Redis-benchmark[10] with the parameters of $c = 200$ and $d = 1\,000$. Memcached workloads are generated by using the load generation tool Memaslap[11] with the parameters of $T = 3$, $c = 810$, $X = 10\,000$, $w = 100\,000$, and $d = 1\,000$.

As can be seen, the existing state-of-the-art SGX-based solution is not accurate enough and also with non-negligible overhead. Nevertheless, we shall also admit its potential in guaranteeing the reliability of performance monitoring. We are therefore motivated to overcome the limitations of SGX-based solution to present a new design with low error rate and overhead, with reliability guaranteed at the same time.

[9] https://hub.docker.com/r/tozd/sgx, May 2022.
[10] https://redis.io/docs/reference/optimization/benchmarks/, July 2022.
[11] http://docs.libmemcached.org/bin/memaslap.html, July 2022.

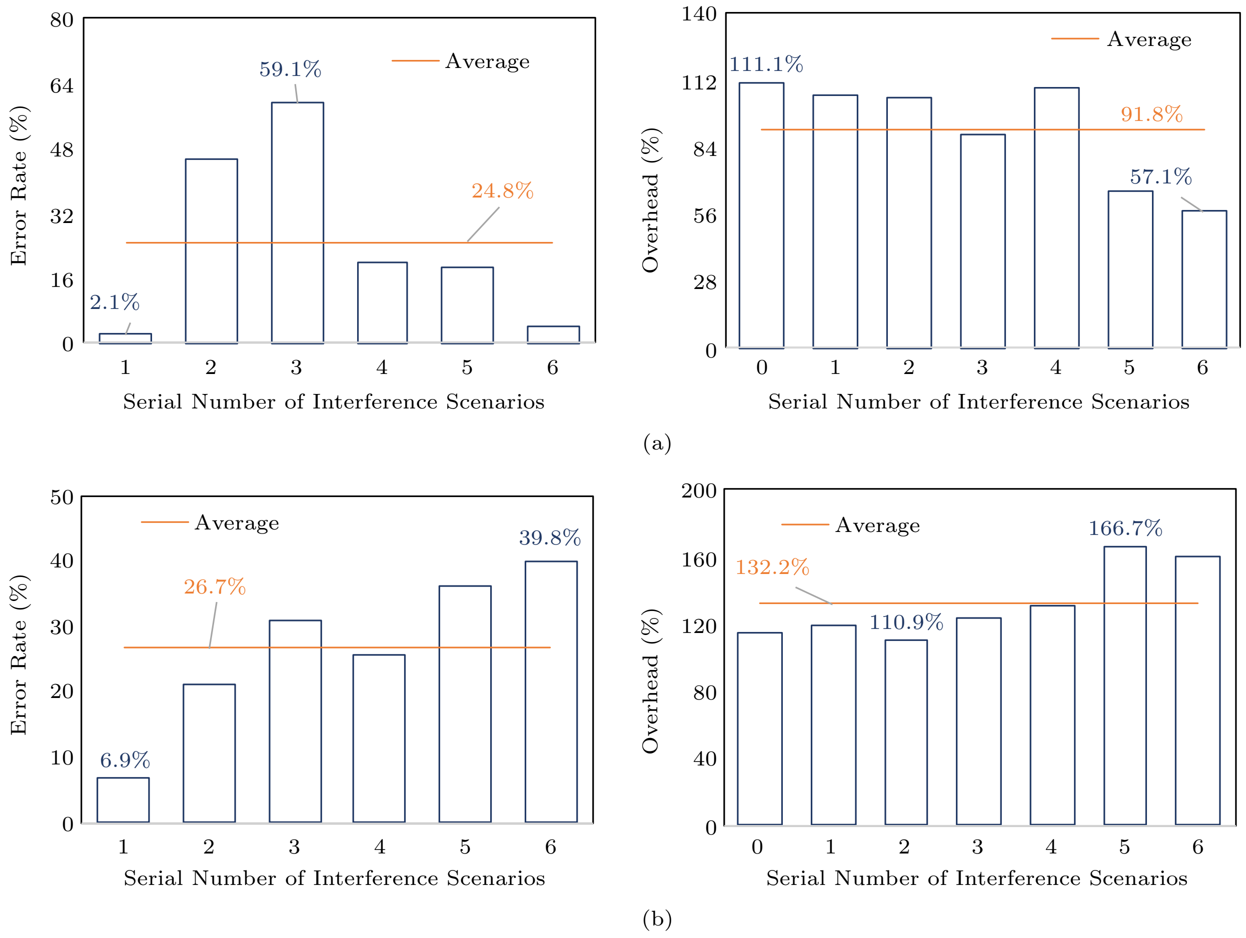


Fig.3. OVERSEE's error rate and overhead. (a) π calculation (CPU-intensive). (b) BubbleSort (memory-intensive).

3.2.3 No Incentive Mechanisms Considering Performance Assessment

For the incentive system in Fig.2, it is expected that an incentive mechanism that can encourage more cloud providers to participate in, and avoid the phenomenon that the bad money drives out the good. Existing cloud platforms usually use the number of historical workloads correctly executed as a metric for reputation rating (e.g., Golem, iExec, and SONM) and asset mortgage (e.g., Akash). Other peer-to-peer computing systems also apply economy-based [9, 10] or reciprocity-based [11–13] methods through integrating the game theory [14, 15] and the auction algorithm [16]. They use the reputation or contribution of the participants as the input, to incentivize the participants to conduct good behaviours.

However, none of them considers computation performance assessment in the incentive mechanisms. Moreover, directly applying existing incentive mechanisms is infeasible, because there are no reliable and verifiable performance monitoring results to be the input, and no suitable performance assessment metric designed for the performance incentive mechanisms. However, obviously the extent that a cloud provider can meet its claimed performance when executing users' workloads shall be an important factor in evaluating its reputation. Without an appropriate incentive mechanism with reliable performance assessment metrics, a cloud provider may have no motivation to provide high performance when executing users' workloads, i.e., not maintaining the stability relative to cloud providers' claimed performance, and may even intentionally claim higher performance beyond their capability to gain more users' workloads for higher revenue. In detail, a malicious cloud provider can make more users' workloads be allocated to it by claiming high performance, but offer low performance to save costs when executing the workloads to gain more profits.

To address this problem, we propose a unified performance assessment metric for cloud providers in the decentralized cloud, and then take this metric into the cloud provider selection decision, to incentivize cloud providers to offer their real performance as claimed.

4 Design of RODE

RODE is designed to first generate reliable and verifiable performance monitoring results. Based on the performance monitoring results, a unified performance assessment metric is formed for cloud provider selection. In this section, we first introduce the performance monitoring mechanism to solve the problems in Subsection 3.2.1 and Subsection 3.2.2, and then the assessment of the claimed performance mechanism (AoCP) to solve the problem in Subsection 3.2.3.

4.1 Performance Monitoring Mechanism

With the consideration of RODE's generality, we integrate the two common characteristics which are included in most of the TEEs to design our mechanism, i.e., secured memory protection and remote attestation. In this subsection, we use Intel SGX (one of the commonly-used TEEs nowadays) to illustrate our design. As stated in Subsection 3.2.2, to avoid the limitations of SGX and improve the efficiency of performance monitoring while guaranteeing reliability and verifiability, instead of putting all the workloads in SGX as OVERSEE does, we offload them outside SGX, and only make use of SGX to hide the critical part of the performance monitoring process. In this way, the cloud provider cannot distinguish the performance monitoring workload from the normal workload. Once the workload is executed, the cloud provider can use the SGX's report to prove the validity of performance results.

As shown in Fig.4, our mechanism supports piecewise performance monitoring. As different code fragments of a workload may have heterogeneous resource consumption characteristics, using different monitoring methods based on the dominant resource shall be preferred. We divide the workload into normal, operation-predictable, and operation-unpredictable code fragments, and classify the monitoring methods accordingly. We first generate a random key pair (PK, SK) in advance. PK encrypts each code fragment's flag, which is an identifier of the code fragment, and embeds the encrypted flag into the workload code. SK (hardcoded into the enclave) decrypts the encrypted

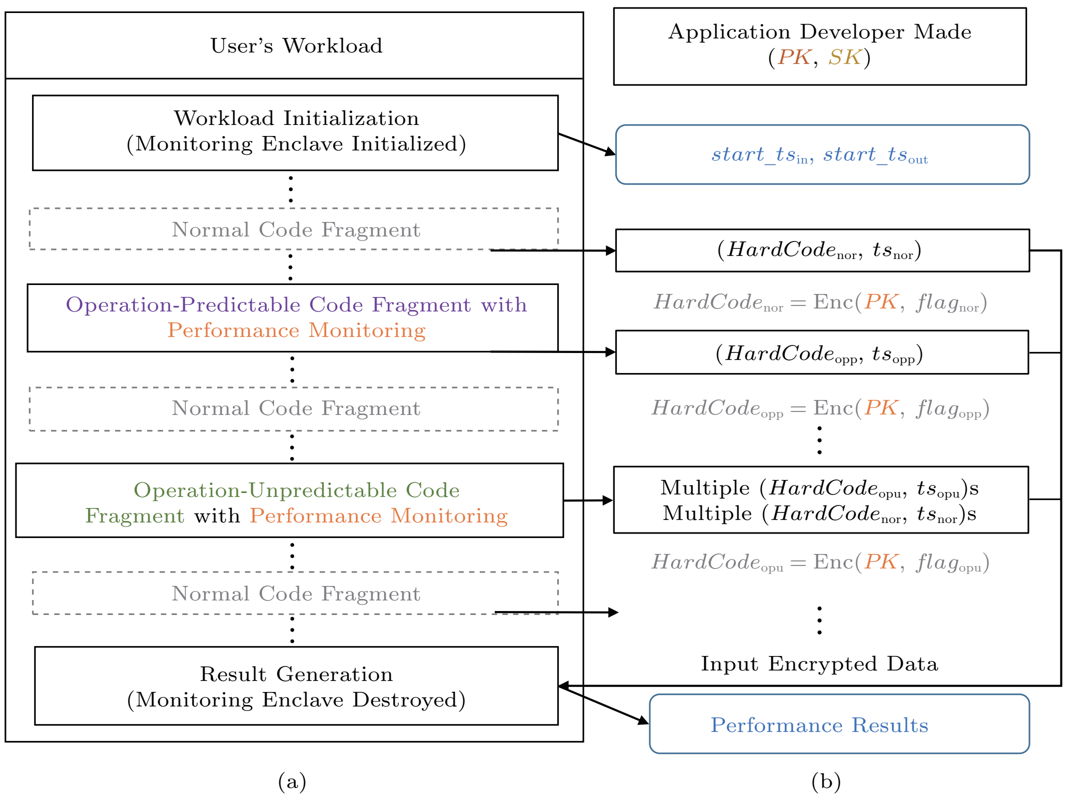

User's Workload
Workload Initialization (Monitoring Enclave Initialized)
Normal Code Fragment
Operation-Predictable Code Fragment with Performance Monitoring
Normal Code Fragment
Operation-Unpredictable Code Fragment with Performance Monitoring
Normal Code Fragment
Result Generation (Monitoring Enclave Destroyed)
Application Developer Made (PK, SK)
$start\_ts_{\rm in}$, $start\_ts_{\rm out}$
($HardCode_{\rm nor}$, $ts_{\rm nor}$)
$HardCode_{\rm nor} = {\rm Enc}(PK, flag_{\rm nor})$
($HardCode_{\rm opp}$, $ts_{\rm opp}$)
$HardCode_{\rm opp} = {\rm Enc}(PK, flag_{\rm opp})$
Multiple ($HardCode_{\rm opu}$, $ts_{\rm opu}$)s
Multiple ($HardCode_{\rm nor}$, $ts_{\rm nor}$)s
$HardCode_{\rm opu} = {\rm Enc}(PK, flag_{\rm opu})$
Input Encrypted Data
Performance Results
(a)
(b)

Fig.4. Overview of the performance monitoring mechanism. (a) Execution process of a user's workload. (b) Generated data within the execution process.

performance monitoring the data passed into the enclave. For each code fragment, we record the hard code that identifies its content and its end timestamp. For example, for a normal code fragment, we record $HardCode_{\rm nor}$=Enc(PK, $flag_{\rm nor}$) and the end timestamp $ts_{\rm nor}$. Enc() represents the encryption function, and $flag_{\rm nor}$ is the plaintext of $HardCode_{\rm nor}$, which can identify that this fragment is a normal code fragment. The purpose of encrypting each code fragment's flag is to avoid malicious cloud providers recognizing and tampering with the performance monitoring timestamps generated in the untrusted memory, and the detailed analysis is in Subsection 4.1.5.

We present the four key components of the performance monitoring mechanism and discuss its reliability as follows.

4.1.1 Workload Initialization

It is responsible for determining the workload's start timestamp reliably. SGX can obtain the trusted system timestamp inside the enclave with the granularity of 1 s, which is not fine enough. Therefore, RODE gets the start timestamp outside the enclave with millisecond granularity and uses the trusted timestamp of Intel SGX as a reference to guarantee its reliability.

A monitoring enclave, as an enclave whose internal code is specifically used for performance monitoring, is initialized during workload initialization. After the initialization, RODE enters the monitoring enclave to generate a trusted start timestamp. The monitoring enclave records the timestamp $start\_ts_{\rm in}$ in the trusted memory, and then returns to the outside part, who can then generate a timestamp $start\_ts_{\rm out}$ with millisecond granularity in the untrusted memory. Since $start\_ts_{\rm out}$ is obtained right after $start\_ts_{\rm in}$, we can guarantee the reliability of $start\_ts_{\rm out}$ by checking whether the difference between the two timestamps is within 1 s (i.e., the granularity of $start\_ts_{\rm in}$) or not.

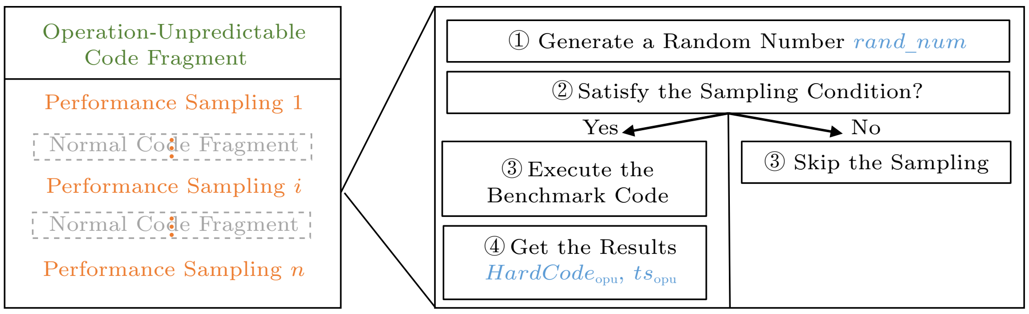
4.1.2 Operation-Predictable Code Fragment with Performance Monitoring

The dominant resource consumption of some code fragments (e.g., a fragment that reads or writes a dataset with a known size) can be predicted accurately (or known exactly). We regard them as operation-predictable code fragments, for which we only need to get the execution time as the metric for performance monitoring. Therefore, for an operation-predictable code fragment, we get the start timestamp (the end timestamp of the previous code fragment), the end timestamp $ts_{\rm opp}$, and the hard code of this fragment $HardCode_{\rm opp}$, which includes the encryption of marking read or write accesses and the dataset size.

4.1.3 Operation-Unpredictable Code Fragment with Performance Monitoring

For most code fragments, their dominant resource consumption cannot be predicted accurately, e.g., CPU-intensive or memory-intensive code fragments. Because of the uncertain numbers of CPU cycles and memory accesses, we cannot make precise performance monitoring with only the execution time. Therefore, we design the performance sampling solution. It requires application developers to insert public micro benchmarks according to their applications' resource consumption characteristics. A developer can select the micro benchmark, which takes a small amount of time (e.g., tens of milliseconds), for each code fragment, and can also decide the frequency of performance samplings. RODE records the execution time values of all inserted benchmarks and uses the average value of them as the metric to determine the code fragment's performance.

As shown in Fig.5, for each performance sampling, it first generates a random number $rand\_num$ to make the occurrence of this sampling to be probabilistic. This step aims at making the sampling positions and the times random no matter when the code fragment is



Operation-Unpredictable Code Fragment
Performance Sampling 1
Normal Code Fragment
Performance Sampling i
Normal Code Fragment
Performance Sampling n
① Generate a Random Number $rand\_num$
② Satisfy the Sampling Condition?
Yes
No
③ Execute the Benchmark Code
③ Skip the Sampling
④ Get the Results $HardCode_{\rm opu}$, $ts_{\rm opu}$

Fig.5. Details of the performance sampling solution for operation-unpredictable code fragments.

executed, so as to prevent a malicious cloud provider from attacking by re-running this workload. If the sampling condition is satisfied, i.e., $rand\_num$ exceeds the threshold set by the application developer, we execute the chosen micro benchmark, and get the results $HardCode_{\text{opu}}$ (the encrypted micro benchmark ID) and the end timestamp ts_{opu}. If it is not satisfied, this sampling is skipped. We can get $rand\_num$ between 0 and 1, and set the sampling threshold as $THR = \frac{S_{\text{exp}}}{S_{\max}}$, where S_{exp} represents the expected sampling frequency and $S_{\max}$ represents the maximum sampling frequency that can be in the code fragment. For instance, if we set $THR = 0.5$ and $S_{\max} = 2$ samplings/s in the code, we can get the expected sampling frequency with $S_{\text{exp}} = 1$ sampling/s.

The performance sampling solution divides this code fragment into multiple smaller code fragments, including the normal code fragments and the performance samplings. Therefore, multiple pairs of $(HardCode, ts)$ of normal fragments and performance samplings can be generated after this performance monitoring, as shown in Fig.4.

For the workloads mixing with predictable and unpredictable code fragments, the application developers are responsible for the classification between them, as they know the resource usage of their workloads. Moreover, they can use performance analysis tools to assist, e.g., PMU events.

4.1.4 Result Generation

Result generation is responsible for: 1) achieving the workload end timestamp reliably; 2) processing the performance monitoring data; 3) generating the performance summaries and the SGX's report.

As shown in Fig. 6, the code outside the enclave firstly gets the end timestamp end_ts_{out} (step 1), and then makes an ECALL to enter the enclave to record the trusted end timestamp end_ts_{in} (steps 2 and 3). Next, the outside code inputs the encrypted data generated during the workload execution into the monitoring enclave (step 4). The monitoring enclave uses the private key SK, which is hard-coded into the enclave code by the application developer, to convert the ciphertext data to the plaintext data (step 5). Then, the monitoring enclave generates the performance monitoring data for each performance monitoring code fragment (step 6), i.e., {Read/Write, Read/Write speed} for the operation-predictable code fragment and {benchmark ID, average execution time} for the operation-unpredictable code fragment. Our mechanism collects all the data to generate the performance summaries (step 7).

SGX provides a CPU-signed data structure called "report" for each enclave, which can be used for remote attestation to confirm the identity of the remote enclave. A report contains the hash of the code inside the enclave, the user-defined data which can be used

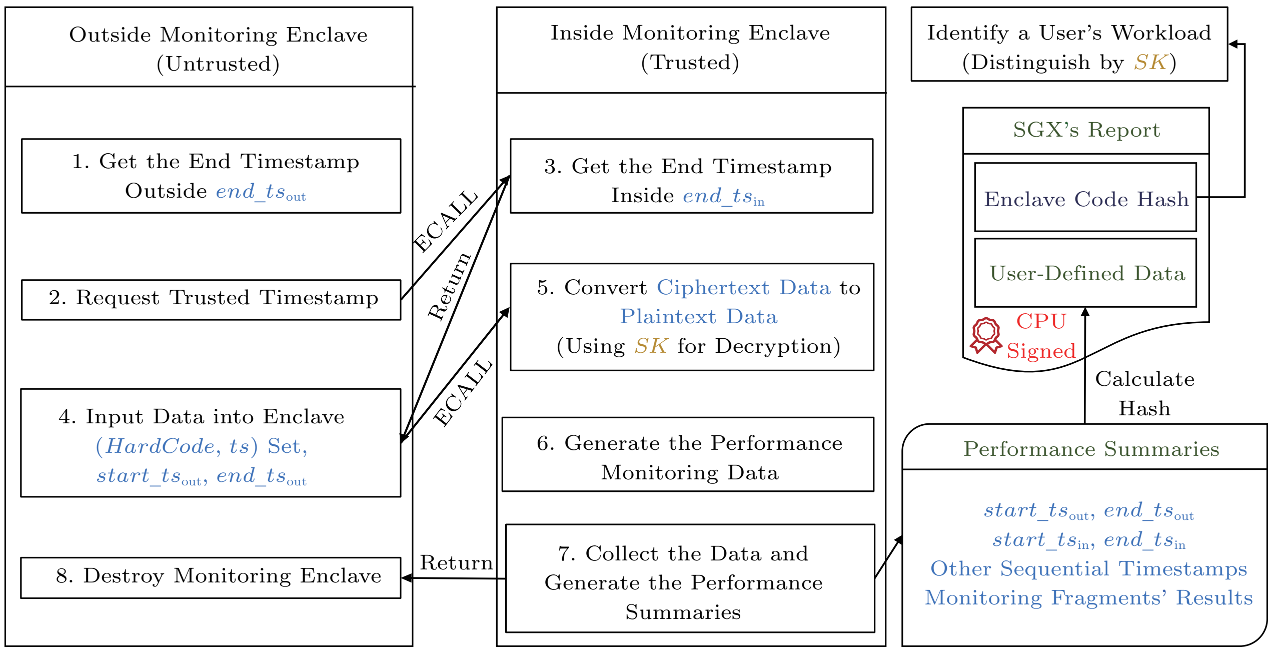

Outside Monitoring Enclave (Untrusted)
1. Get the End Timestamp Outside end_ts_{out}
2. Request Trusted Timestamp
4. Input Data into Enclave $(HardCode, ts)$ Set, $start\_ts_{\text{out}}$, end_ts_{out}
8. Destroy Monitoring Enclave
ECALL
Return
ECALL
Return
Inside Monitoring Enclave (Trusted)
3. Get the End Timestamp Inside end_ts_{in}
5. Convert Ciphertext Data to Plaintext Data (Using SK for Decryption)
6. Generate the Performance Monitoring Data
7. Collect the Data and Generate the Performance Summaries
Identify a User's Workload (Distinguish by SK)
SGX's Report
Enclave Code Hash
User-Defined Data
CPU Signed
Calculate Hash
Performance Summaries
$start\_ts_{\text{out}}$, end_ts_{out}
$start\_ts_{\text{in}}$, end_ts_{in}
Other Sequential Timestamps
Monitoring Fragments' Results

Fig.6. Detailed process of result generation. The finally generated performance results include the performance summaries and the SGX's report.

by application developers, and some other fields. As shown in the right part of Fig.6, when the performance summaries are generated, our mechanism calculates the hash value of them and fills this value into the user-defined data of the report. Hence the final results of the performance monitoring mechanism include the Intel SGX's report and the performance summaries. In addition, we use the enclave code hash in SGX's report to identify different users' workloads based on the difference of the hardcoded private key SK. At last, the monitoring enclave is destroyed (step 8).

It is worth noting that most of the other TEEs also have the data structure similar to Intel SGX's report, e.g., AMD SEV-SNP's Attestation Report [21], which can be used to conduct the same operations for generating performance results for RODE.

4.1.5 Reliability Analysis

As the cloud providers can only get the executable files of users' workloads, they have to conduct reverse engineering to convert the binary code into the source code, for distinguishing performance monitoring instructions from the original workload instructions. However, reverse engineering [34–36] is non-trivial, time-consuming, and resource-consuming. Even though the cloud providers can find out which parts are used for performance monitoring, the overhead incurred even overwhelms the benefit that can be obtained. Hence, it is hard for cloud providers to cheat by speculating the monitoring instructions from the executable files, and then increasing their performance dynamically to respond to each monitoring instruction. Similarly, it is also impractical to replace the monitoring benchmarks in the executable files to get better monitoring results.

One may notice that malicious cloud providers may cheat by forging the timestamps generated in the untrusted memory. To prevent such cheating, we use PK to encrypt each code fragment's flag, making malicious cloud providers unable to recognize or tamper with performance monitoring timestamps (mixed with $ts_{\rm nor}$s) generated in the untrusted memory. Moreover, it is also infeasible for cloud providers to directly make all code fragments' execution time tiny, because the workload execution time is achieved reliably through four timestamps $start\_ts_{\rm out}$, $start\_ts_{\rm in}$, $end\_ts_{\rm out}$ and $end\_ts_{\rm in}$. Furthermore, before the execution ends, cloud providers cannot know which timestamp is $end\_ts_{\rm out}$, whose validity can be verified by $end\_ts_{\rm in}$. Therefore, tampering with timestamps can cause the time difference between $end\_ts_{\rm out}$ and $end\_ts_{\rm in}$ more than 1 s, and malicious behaviors can be detected.

RODE offers a dynamic link library for the application developers to implement reliable performance monitoring. The application developer only needs to call the library functions with few modifications to the source code. Besides, the accuracy of performance monitoring is related to the monitoring positions and sampling frequency set by the application developer, which are the input parameters of the dynamic link library calling.

4.2 Assessment of the Claimed Performance Mechanism

Supported by the performance monitoring mechanism, the purpose of AoCP is to generate a unified metric to assess the performance of a cloud server executing a user's workload in the decentralized cloud. Then, the unified metric can be the reliable input to the incentive mechanism. We divide the process of AoCP into three steps: 1) verifying the performance results; 2) comparing the monitoring performance with the claimed performance of the corresponding cloud provider's cloud server, and updating the unified performance assessment in the metric of stability value (SV); 3) including the performance assessment metric into cloud provider selection for users workloads to incentivize cloud providers to offer better performance. At the end of this subsection, we also analyze the differences and similarities between AoCP and existing peer-to-peer incentive mechanisms in depth.

4.2.1 Verification of Performance Results

Based on the performance monitoring mechanism, a cloud provider can obtain the reliable and verifiable performance results of a user's workload, including the performance summaries and the SGX's report with the CPU signature generated by SGX. The verification process is shown in Algorithm 1.

We first need to verify the validity of the SGX's report to confirm the identity of a remote enclave by using Intel Attestation Service (IAS) (lines 1–4 in Algorithm 1). Once the SGX's report is validated, we can trust the user-defined data embedded in it, which contains the hash value of all the performance summaries written by the monitoring enclave, as mentioned in the right part of Fig.6. As shown in lines 5–9, we calculate the hash value of the performance summaries, and check whether it is consistent with the user-defined field in the SGX's report. This step prevents cloud providers from generating inveracious performance results. Then, we get all the timestamps generated during workload execution from the performance summaries, and verify whether $start\_ts_{\rm in}$ is next to $start\_ts_{\rm out}$, $end\_ts_{\rm out}$ is next to $end\_ts_{\rm in}$, and the other timestamps are ordered in time, as shown in lines 10–16. This step prevents the malicious cloud providers from cheating by tampering with timestamp data. After the above verification process, the validity of the performance results is proved. The reliable performance results can be seen as the basic input of the AoCP mechanism.

Algorithm 1. Verification of Performance Results

```
 1: $flag_{\rm sig}$=RemoteAttestationbyIAS(SGX's report)
 2: if $flag_{\rm sig} \neq$ TRUE then
 3:     return FALSE
 4: end if
 5: $Hash_{\rm Perf} \longleftarrow$ CalHash(performance summaries)
 6: $Hash_{\rm SGX} \longleftarrow$ GetUserDefined(SGX's report)
 7: if $Hash_{\rm Perf} \neq Hash_{\rm SGX}$ then
 8:     return FALSE
 9: end if
10: $start\_ts_{\rm out}, start\_ts_{\rm in}, ts_1, ..., ts_n, end\_ts_{\rm out}, end\_ts_{\rm in}$
    $\longleftarrow$ GetTimestamps(performance summaries)
11: if $start\_ts_{\rm out} - start\_ts_{\rm in} \geqslant 1$ or
    $|end\_ts_{\rm in} - end\_ts_{\rm out}| \geqslant 1$ then
12:     return FALSE
13: end if
14: if exists($k \in [1, n]$ and $ts_k \geqslant ts_{k+1}$) then
15:     return FALSE
16: end if
17: return TRUE
```

It is worth noting that most of the TEEs also offer the remote attestation service similar to Intel IAS, e.g., AMD-SP firmware[21]. The time complexity of Algorithm 1 is $O(n)$, and its execution time is about 190 ms.

4.2.2 Updating Performance Assessment Metric

Each cloud provider can claim the performance of its cloud servers, but cannot always achieve the claimed performance when executing a user's workload. To this end, we use $Rate_i$ to quantify the degree that a cloud server achieves its claimed performance for executing a user's workload i as follows:

$$Rate_i = \sum_{j=1}^{m} \left(weight_j \times \frac{PerfMonitor_j}{PerfClaimed_j} \right),$$

where $PerfMonitor_j$ is the j-th performance monitoring result for the user's workload. $PerfClaimed_j$ is the corresponding claimed performance (e.g., benchmark execution results claimed by the cloud provider). $weight_j$ indicates the significance of the j-th performance monitoring among all the workload's monitoring, and is set according to the workload resource consumption characteristics by the application developer. In addition, we have $weight_j \in [0, 1]$ and $\sum_1^m weight_j = 1$, where m is the total number of performance monitoring in this workload. For each user's workload, we calculate the weighted average ratio of all the performance monitoring results to the cloud server's claimed performance as $Rate_i$. Based on $Rate_i$, we further define the stability value (SV) as

$$SV_{\rm new} = SV_{\rm old} \times \frac{1}{n} \sum_{i=1}^{n} Rate_i,$$

to measure the stability that a cloud server can achieve its claimed performance in each workload execution round. n represents the number of workloads in each round, and n can be tens to thousands of workloads depending on the scale of the decentralized cloud platform. The initial SV of each cloud server is 1 and SV is in the range of $[0, 1]$. For each round, we calculate the average $Rate_i$ of all the workloads, and multiply it by the old SV value $SV_{\rm old}$ to get the new SV value $SV_{\rm new}$. If the newly updated value is larger than 1, we set $SV = 1$ to indicate that the claimed performance is achieved on average for this round's workloads. Therefore, the claimed performance and SV form the unified metric to jointly assess the actual performance of a specific cloud server from a cloud provider.

4.2.3 Including Performance Assessment Metric into Cloud Provider Selection

Based on the performance assessment results generated by AoCP, a user can select the most appropriate cloud server for its workloads according to its preferences. We design a cloud provider selection policy as follows. We first calculate the total score $score_{\rm Total}$ of each cloud server for the workload to be executed, based on the workload's resource demands and the performance assessment metric of the cloud servers. Then, the selected cloud server executes the workload and updates its SV. $score_{\rm Total}$ is defined as follows:

$$\begin{aligned} score_{\rm Total} = {} & \sum_{i=1}^{n} (wt_i \times score_i) + \\ & wt_{SV} \times score_{SV}, \end{aligned} \tag{1}$$

where $score_i$, ranging from 0 to 100, rates the latest claimed performance of a cloud server to execute benchmark i, and $score_{SV}$ rates the performance stability with $score_{SV} = SV \times 100$; wt_i and wt_{SV}, ranging from 0 to 1, rate the similarity of benchmark i with the workload in terms of resource demand, and rate the stability value (SV) demand of satisfying the claimed performance during the workload execution, respectively. In addition, it shall also satisfy constraints $\sum_{i=1}^{n}(wt_i)+wt_{SV}=1$ and $0 \leqslant wt_i \leqslant 1, 0 \leqslant wt_{SV} \leqslant 1$. As application developers can know the resource and SV demand of the workload based on historical data, they can determine wt_i and wt_{SV}. Generally, there are three kinds of benchmarks, i.e., CPU-intensive, memory-intensive, and IO-intensive.

In this policy, to obtain the changes of cloud providers' performance stability relative to their claimed performance in a timely manner, we use recent several rounds of workload submissions to update their SVs (i.e., $score_{SV}$), e.g., we use three rounds in the experiments of Subsection 5.4. In this way, for the cloud provider who cheats for long and has been assessed with low SV, we can avoid the cloud provider having no motivation to provide real claimed performance and maintain the performance stability in the future. In other words, as long as the cloud provider begins to maintain its performance stability (relative to its claimed performance) in the recent workload submission rounds, its SV will be steadily improved, and thus will get more users' workloads in subsequent rounds.

The cloud provider selection policy can be seen as AoCP's incentive scheme that has the reliable performance assessment metric as the input. We mainly use this policy to prove the incentive effect of our performance assessment metric and we can also integrate other peer-to-peer incentive schemes into our AoCP mechanism. In Subsection 5.4, we use this selection policy to prove that the cloud providers with better performance assessment metrics of AoCP can achieve more workloads, and thus enforce the cloud providers to offer the performance as claimed to make profits.

To enable decentralized, reliable, and transparent performance assessment metric updating and cloud provider selection, RODE implements AoCP with the Blockchain technology. The Blockchain technology is the general technique of decentralized cloud platforms, and the implementation of AoCP does not need to rely on any customized attributes of a specific decentralized cloud's Blockchain system. Therefore, we implement AoCP by developing smart contracts on Ethereum to validate its simplicity and feasibility. In detail, we develop an update contract to update cloud providers' claimed performance, a reference contract to declare the workloads to be executed by a cloud provider, and a performance contract to update the cloud providers' SVs based on the results from the performance monitoring mechanism. The integration of the Blockchain Ethereum also brings great improvement to get rid of the centralization mode of most of the incentive mechanisms of peer-to-peer computing systems.

4.2.4 Comparisons Between AoCP and P2P Incentives

In this subsection, we conduct comparisons between RODE's AoCP mechanism and existing peer-to-peer incentive mechanisms in depth. The analysis is conducted from three aspects and summarized in Table 4.

Firstly, since the decentralized cloud providers are untrusted, it is of significance to obtain reliable cloud providers' performance data for the incentive mechanism. AoCP uses the performance results generated by the performance monitoring mechanism (Subsection 4.1) as the basic input to the performance incentive mechanism, and designs the corresponding verification method (Subsection 4.2.1) to make the input to be reliable and verifiable. By contrast, existing peer-to-peer incentive mechanisms do not consider the reliability of the input [9–13].

Secondly, the AoCP mechanism customizes the performance stability value (SV in Subsection 4.2.2) as the metric for the performance incentive mechanism. This metric can effectively assess whether cloud providers provide the performance as they claimed, encourage cloud providers to maintain their performance stability, and incentivize cloud providers to present real claimed performance. By contrast, although existing peer-to-peer incentive mechanisms have similar metrics (e.g., peers' contribution [11] or reputation [13]), they have no corresponding design for performance assessment, thereby their metrics cannot be used directly.

Thirdly, AoCP designs an SV-based cloud provider selection policy as the incentive scheme to incentivize cloud providers to provide the performance they claimed. Other existing peer-to-peer incentive mechanisms use two categories of incentive schemes:

Table 4. Comparisons Between AoCP and Peer-to-Peer Incentive Mechanisms

Incentive Mechanism	Input	Assessment Metric	Incentive Scheme
AoCP mechanism	Reliable	Performance stability value (SV)	SV-based cloud provider selection
Peer-to-peer incentive mechanisms [9–13]	Unreliable	No suitable metric	Economy-based [9, 10]
			Reciprocity-based [11–13]

economy-based [9,10] and reciprocity-based [11–13], and include various theories and algorithms, e.g., game theory [14,15] and double auction [16]. They utilize the contribution or reputation of peers as the input to avoid the malicious behaviors, which have a similar idea to AoCP. We can also integrate other incentive schemes of peer-to-peer computing into AoCP's incentive scheme. In addition, AoCP also considers some situations that may occur especially in the scenario of performance assessment incentives, and designs some corresponding solutions in Subsection 4.2.3.

5 Evaluation of RODE

In this section, we conduct both testbed experiments (Subsection 5.2 and Subsection 5.3) and emulation experiments (Subsection 5.4), to evaluate the feasibility of the performance monitoring mechanism and the effectiveness of the AoCP mechanism, respectively.

5.1 Evaluation Setup

Testbed Experiments. The purpose of the testbed experiments is to verify that our performance monitoring mechanism can effectively reflect the trend of the performance changes of users' workloads. Specifically, we check whether the micro benchmarks in our performance sampling solution have the same performance change trend with the workloads. Since current benchmarks usually take a long execution time, we extract the logic of the SPEC CPU[12] and Stream[13] benchmarks to build the micro benchmarks for our experiments that complete within tens of milliseconds.

Since other performance monitoring methods of the decentralized cloud (stated in Subsection 3.2.1) cannot guarantee reliability, RODE is fundamentally stronger than them. Instead, we compare RODE with the state-of-the-art work OVERSEE [7] which also uses TEE techniques to address the reliable performance monitoring problem in an environment of mutual distrust. We implement RODE's performance sampling solution, and apply the same workloads and benchmarks (shown in Table 2) as in Subsection 3.2.2. Since SGX's minimum timestamp acquisition granularity is 1 s[14] and resource contention does not change too much within a few seconds, we set the performance sampling frequency as one sampling every two seconds for the experiments. The same hardware and software specifications and interference scenarios (shown in Table 1 and Table 3) are used, as reported in Subsection 3.2.2.

Since OVERSEE [7] (executing the entire workload and performance monitoring inside SGX) has the limitations for conducting large workloads, we only use the simple workloads (π calculation and BubbleSort) to compare RODE with it in Subsection 5.2. To fully validate RODE's practicability, we evaluate RODE with large workloads (SVM Train and WordCount) under a dynamic cloud environment in Subsection 5.3.

In the testbed experiments, we define the error of performance monitoring as $error = |1 - \frac{tb_i}{tb_0} / \frac{tw_i}{tw_0}|$, where tb_i and tw_i represent the benchmark and the workload execution time, respectively. Specifically, tb_0 and tw_0 represent the benchmark time and the workload time when there is no interference workload respectively.

Emulation Experiments. The purpose of emulation experiments is to test how AoCP impacts the behaviors of both the cloud providers and users in the decentralized cloud, and then prove that AoCP can incentivize cloud providers to offer performance the same as their claimed one. As far as we know, there are no available public traces of decentralized cloud platforms for us to conduct the experiments. Thus, we have to generate the traces by ourselves in a rational manner to better verify the effectiveness of the AoCP mechanism.

With considerations of the experiment purpose, the emulation traces should be related to cloud providers' behaviors, but independent of the workload submission behaviors or cloud providers' initial performance advantages. Therefore, to control irrelevant variables, we need to submit workloads with identical quantities in each round, make each type of workloads (CPU-, memory-, and IO-intensive) be the same amount in each round, and make each cloud provider have the same performance advantages when they all behave well (i.e., cloud providers can be allocated an equal amount of workloads in each round if they do not reduce the performance as they claimed). Moreover, to explore the impact of different resource and SV demand of (1) on workload allocations, we need to set several different sets of wt_i and wt_{SV} for each type of workloads. Based on these settings, we emulate different behaviors of cloud providers for multiple workload submission rounds, i.e., not maintaining the performance stability or presenting the real claimed performance. The workload submission round can be seen as a fixed

[12] https://www.spec.org/cpu2017/, May 2022.

[13] https://github.com/jeffhammond/STREAM, May 2022.

[14] https://download.01.org/intel-sgx/sgx-linux/2.11/docs/, May 2022.

time interval, and the experiments can be performed in arbitrary rounds. Since we want to show the process of cloud providers' behaviors affecting the workload allocations, we use the minimum rounds (eight rounds) which can achieve this in the experiments.

In detail, we emulate the process of eight rounds of workload submissions, submit five CPU-intensive workloads (SVM-Train), five memory-intensive workloads (WordCount), and five IO-intensive workloads (MongoDB with 10^5 insertions) in each round, and set different weight parameters to the workloads based on (1) to represent different resource and SV demands, as shown in Table 5. Moreover, we use interference workloads on three high-performance servers in Table 1, to emulate three cloud providers denoted as A, B, and C with different performance respectively. We assume that each cloud provider has one cloud server and each cloud server can provide higher performance to execute one of the three types of workloads than the others (i.e., having the same performance advantages for all workloads). We use float calculation benchmark, array copy (2D) benchmark, and insertions/s of MongoDB to measure the CPU, memory, and IO performance of the cloud providers, respectively, and convert them into performance scores, as shown in Table 6.

In Subsection 5.4, we emulate different situations of cloud providers maintaining performance stability values and claiming performance to verify the effectiveness of the AoCP mechanism.

5.2 Comparison with Baseline

We first evaluate the error and overhead of both schemes. The results are reported in Fig.7, where we can see that RODE's performance sampling method has low error and overhead for both the CPU-intensive and memory-intensive workloads. RODE achieves the error rate of 0.3%–4.8% (2.4% on average) with overhead of 1.22%–2.36% (1.72% on average) for π calculation, and the error rate of 3.0%–8.1% (5.3% on average) with overhead of 0.76%–2.22% (1.34% on average) for BubbleSort. Compared with the results of OVERSEE shown in Fig.3 (see Subsection 3.2.2), RODE decreases the performance monitoring error rate by 77.5% on average and reduces the overhead by 98.4% on average.

The results indicate that RODE can effectively reflect the performance change trend of the workloads. Different from OVERSEE, RODE integrates encryption methods to confuse the performance monitoring fragments with the normal code fragments and only makes use of SGX to hide the critical part of performance monitoring. Moreover, RODE uses the timestamp with millisecond granularity outside SGX's enclave, instead of SGX's trusted timestamp with second granularity, to measure the performance, and thus decreases the performance monitoring error. On the other hand, RODE offloads the workload outside of SGX's enclave and thus reduces both the SGX's overhead and the sampling overhead.

5.3 Practicability of Performance Monitoring Mechanism

In order to validate the practicability of our performance monitoring mechanism under dynamic resource sharing of the cloud environment, we also test the common cloud workloads SVM Train and WordCount (shown in Table 2) in different interference scenarios on the high-performance server as specified in Table 1.

In the experiments, we divide SVM Train into one operation-unpredictable code fragment (CPU-intensive model training) and two operation-predictable code fragments (IO-intensive data reading and trained model writing), and insert one sampling every two seconds

Table 5. Weight Parameters of Workloads in Emulation Experiments

Workload	SVM Train (CPU-Intensive)	WordCount (Memory-Intensive)	MongoDB (IO-Intensive)
wt_{CPU}	$i/(i{+}2)$	0	0
wt_{memory}	0	$i/(i{+}2)$	0
wt_{IO}	0	0	$i/(i{+}2)$
wt_{SV}	$2/(i{+}2)$	$2/(i{+}2)$	$2/(i{+}2)$

Note: Variable i is an integer in the range of [1, 5], which represents a workload with the specific weight parameters.

Table 6. Claimed Performance of Three Cloud Providers in Emulation Experiments

Cloud Provider	$score_{\mathrm{CPU}}$ (π Calculation)	$score_{\mathrm{memory}}$ (Array Copy (2D))	$score_{\mathrm{IO}}$ (insertions/s)
A	100.0 (43.6 ms)	78.9 (17.0 ms)	82.1 (2 273/s)
B	81.0 (53.8 ms)	100.0 (13.4 ms)	77.6 (2 146/s)
C	76.5 (57.0 ms)	81.6 (16.5 ms)	100.0 (2 770/s)

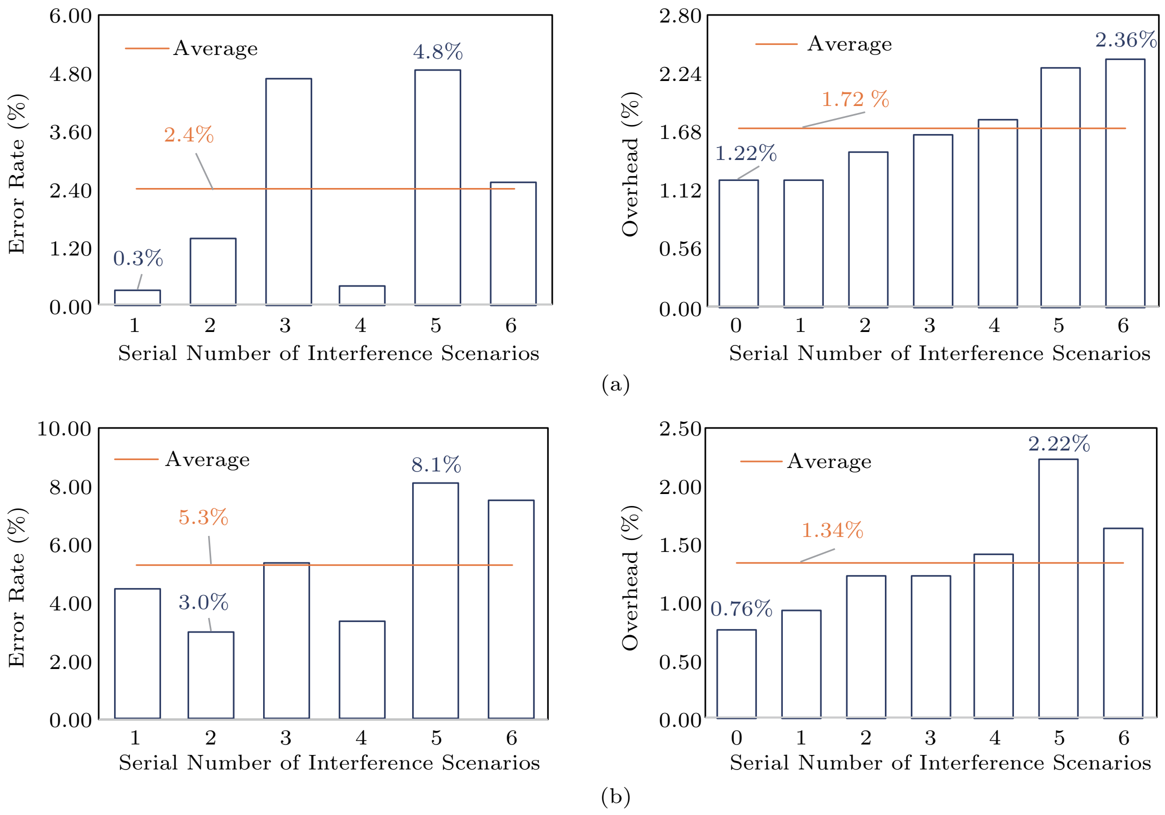


Fig.7. RODE's error rate and overhead. (a) π calculation (CPU-intensive). (b) BubbleSort (memory-intensive).

with Float Calculation benchmark into the CPU-intensive code fragment. Moreover, we divide WordCount into one operation-unpredictable code fragment (memory-intensive word frequency statistics) and two operation-predictable code fragments (IO-intensive data reading and result writing), and insert one sampling every two seconds with Array Copy (2D) benchmark into the memory-intensive code fragment. We conduct the experiments with eight different interference scenarios (one workload for each scenario), as shown in Table 7. The results are shown in Fig. 8. Fig.8(a) and Fig.8(c) show the performance samplings' change trend with the change of interference scenarios of SVM Train and WordCount, respectively. Normalized load (the red line) is the ratio calculated by $\frac{tw_i}{tw_4}$, where tw_i represents the workload execution time in scenario i. The blue dashed lines represent the average benchmark execution time in each scenario, which has a similar change trend to the normalized load. Fig.8(b) and Fig.8(d) show the performance monitoring results of different code fragments (CPU-, memory-, or IO-intensive) of SVM Train and WordCount, respectively. The red curves represent the error of the performance samplings in operation-unpredictable code fragments.

Table 7. Interference Scenarios in Dynamic Experiments

Scenario	SVM Train	WordCount
1	11×(CNN+RL+K-means)	Redis (15 ports)+ Memcached (2 ports)
2	14×(CNN+RL+K-means)	Redis (25 ports)+ Memcached (4 ports)
3	18×(CNN+RL+K-means)	Redis (30 ports)+ Memcached (5 ports)
4	22×(CNN+RL+K-means)	Redis (35 ports)+ Memcached (6 ports)
5	19×(CNN+RL+K-means)	Redis (35 ports)+ Memcached (5 ports)
6	16×(CNN+RL+K-means)	Redis (30 ports)+ Memcached (4 ports)
7	13×(CNN+RL+K-means)	Redis (20 ports)+ Memcached (3 ports)
8	10×(CNN+RL+K-means)	Redis (10 ports)+ Memcached (1 port)

Note: CNN is short for convolutional neural network training. RL is short for reinforcement learning. Redis workloads are generated by using the load generation tool Redis-benchmark with the parameters of $c = 1\,000$ and $d = 8\,000$. Memcached workloads are generated by using the load generation tool Memaslap with the parameters of $T = 3$, $c = 810$, $X = 10\,000$, $w = 100\,000$, and $d = 1\,000$.

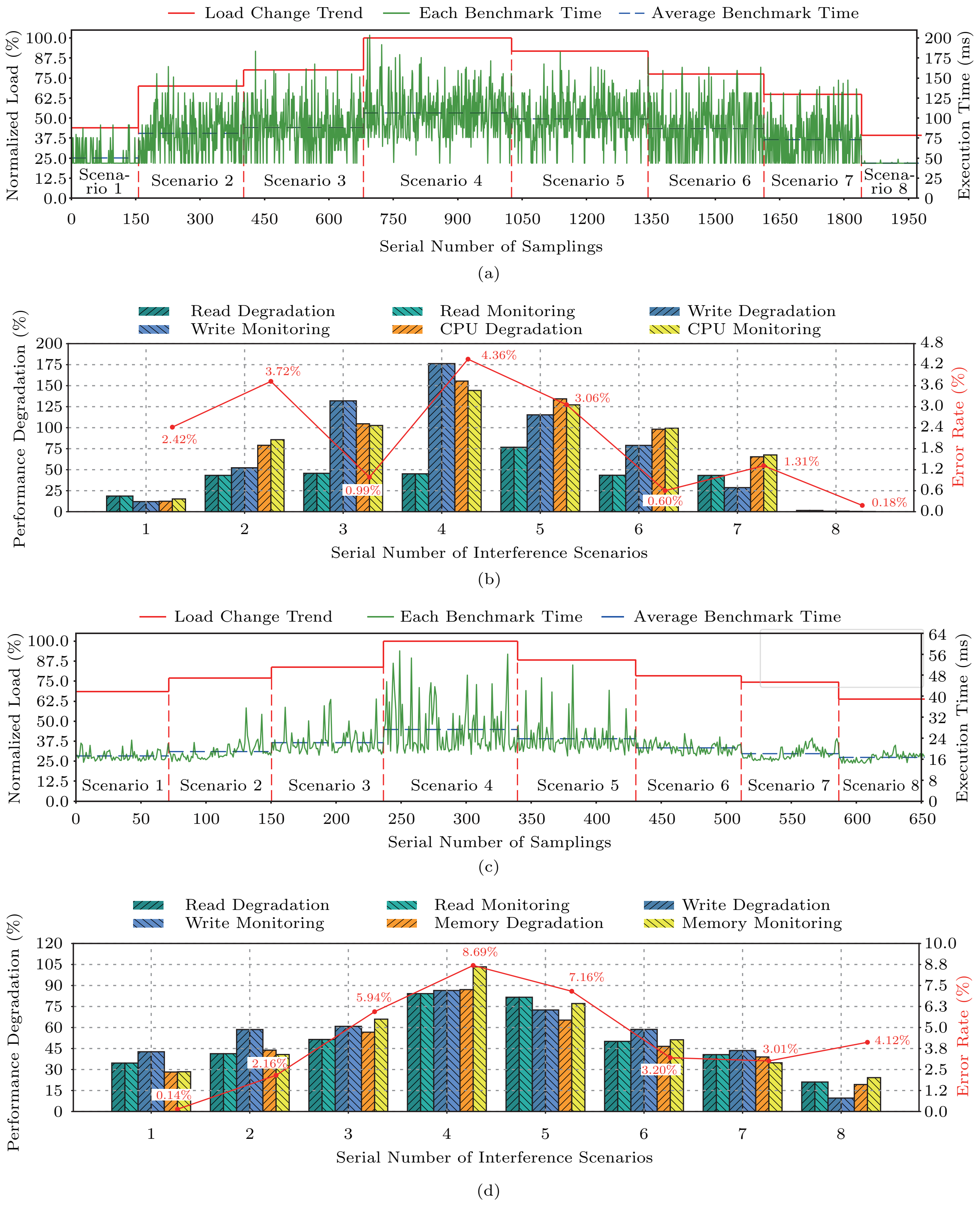


Fig.8. Validation of the performance monitoring mechanism under dynamic resource sharing of the cloud environment. (a) Performance samplings' change trend of SVM Train. (b) Performance monitoring results of SVM Train. (c) Performance samplings' change trend of WordCount. (d) Performance monitoring results of WordCount.

Since the IO-intensive code fragments apply operation-predictable performance monitoring, the monitored performance degradation is equal to the actual performance degradation all the time. For the CPU-intensive code fragment of SVM Train, the monitored performance degradation is almost the same as the actual performance degradation, with the error rate of 0.2%–4.4% (2.1% on average). For the memory-intensive code fragment of WordCount, the monitored performance degradation is slightly larger than the actual performance degradation, with the error of 0.1%–8.7% (4.3% on average). This is because the benchmark is a little more sensitive to the resource contention than the normal workload in general. Nevertheless, the low error validates the practicability of our mechanism.

5.4 Effectiveness of AoCP

5.4.1 *Maintaining Performance Stability Values (SVs)*

We first verify that AoCP can incentivize cloud providers to maintain their performance stability values (SVs). In the experiments, we use the workloads in recent three rounds to update SV, in the consideration of timeliness. To emulate the situation that cloud providers cannot achieve their claimed performance, i.e., cannot maintain their SVs, we make A and B's performance drop for a few rounds by using interference workloads, during which 60% of their workloads experience 30% performance degradation. For each workload, the cloud provider with the highest $score_{\text{Total}}$ can get it. The workload allocation results of all the eight rounds are shown in Fig.9(a). In this figure, "normal" represents the number of workloads executed with the claimed performance in each round, and "slow" represents the number of workloads executed slower than the claimed performance in each round.

We can observe that A fails to meet its claimed performance when processing eight workloads in rounds 2–4 (the sum of the numbers in the row marked "slow" of A in rounds 2–4), leading to the decrease of its $score_{SV}$ in rounds 3–7 (the numbers in the row marked "$score_{SV}$" of A in rounds 3–7). This further leads to

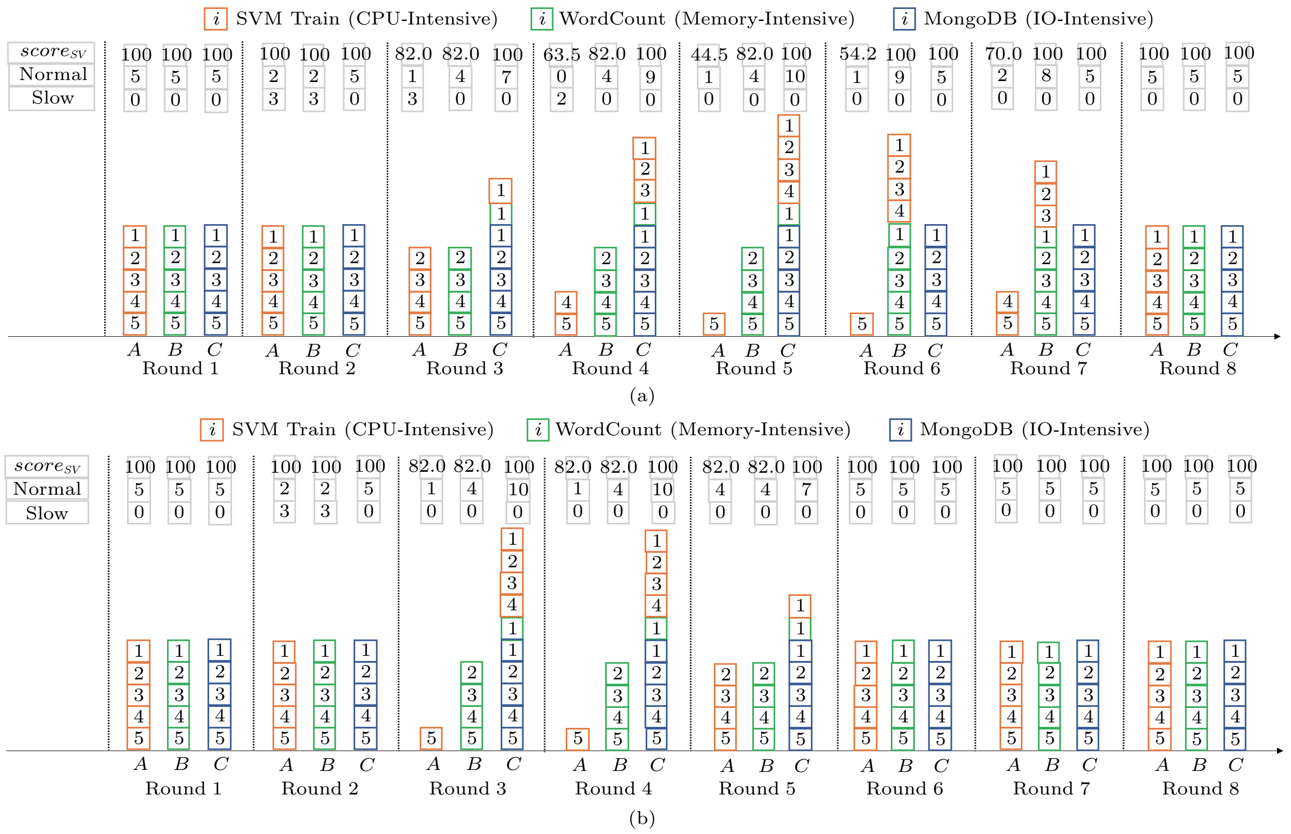


Fig.9. Allocation of different workloads with the changing of SV or claimed performance. (a) The case that cloud providers claim the high performance all the time while it can be lower at times. (b) The case that cloud provider A notifies its performance degradation on time. The i values (1 to 5) inside the colored squares represent different weight parameters for the workloads corresponding to Table 6, e.g., the orange square marked with 1 represents the SVM Train workload with $wt_{\text{CPU}} = 1/3$, $wt_{\text{memory}} = 0$, $wt_{\text{IO}} = 0$, and $wt_{SV} = 2/3$.

the decrease of its $score_{\text{Total}}$ (based on (1)) for the CPU-intensive workloads in these rounds. During these five rounds, some of the CPU-intensive workloads are allocated to B and C, though A still keeps claiming the highest $score_{\text{CPU}}$ as usual. Similarly, B also fails to meet its claimed performance when processing three workloads in round 2. Thus, its $score_{SV}$ and $score_{\text{Total}}$ decrease for three rounds, and some memory-intensive workloads are scheduled to C. Thanks to SV, some users' workloads are appropriately allocated to other cloud providers, and this confirms the effectiveness of AoCP. As the final statistics shown in Table 8, cloud providers A, B, and C get 25, 44, and 51 workloads in total, respectively. It indicates that AoCP can incentivize the cloud providers to maintain their performance stability value (SV).

Table 8. Final Statistics of the Experiment in Fig.9(a)

Cloud Provider	Normal Rate	Total Workload Number
A	$17/25 = 68.0\%$	25
B	$41/44 = 93.2\%$	44
C	$51/51 = 100\%$	51

Under AoCP's incentive on maintaining performance stability value (SV), if $score_{SV}$s of all cloud providers maintain at 100 during the eight rounds, the workloads' execution time can reduce by 10.9% in total compared with the situation in Fig.9(a), where the time for SVM Train and WordCount can be reduced on average by 14.5% (21.1% in the best round) and by 3.7% (15.3% in the best round), respectively. MongoDB workload's performance keeps stable because no performance degradation occurs in the situation of Fig.9(a).

5.4.2 *Presenting Real Claimed Performance*

In order to check whether AoCP can incentivize the cloud providers to honestly report their performance, we conduct another emulation experiment with the same cloud providers and users' workloads, and show the results in Fig.9(b). The only difference is that cloud provider A actively decreases its claimed performance at rounds 3 and 4. In detail, A decreases its $score_{\text{CPU}}$ to 85, $score_{\text{memory}}$ to 63.9, and $score_{\text{IO}}$ to 67.1. A honestly reports its real performance information in order to avoid the decrease of $score_{SV}$.

As observed, although A gets four fewer workloads in the situation of Fig. 9(b) than in the situation of Fig.9(a) in rounds 3 and 4, it can quickly get the CPU-intensive workloads back as soon as its performance recovers. As the final statistics shown in Table 9, cloud provider A gets more workloads (31 workloads in total) in Fig.9(b) than in Fig.9(a), indicating that AoCP can incentivize the cloud providers to honestly present their real performance information.

Table 9. Final Statistics of the Experiment in Fig.9(b)

Cloud Provider	Normal Rate	Total Workload Number
A	$28/31 = 90.3\%$	31
B	$34/37 = 91.9\%$	37
C	$52/52 = 100\%$	52

With AoCP's incentive on presenting real performance, the workloads' execution time can be reduced by 4.2% in total in the situation of Fig.9(b) compared with the situation of Fig. 9(a), where the execution time of SVM Train can be reduced by 6.2% on average (16.4% in the best round). The time for WordCount and MongoDB does not reduce because there is no change for them in the situation of Fig.9(b).

5.4.3 *Large-Scale Simulation*

We also conduct simulations to expand the scale of the experiments, with the process of eight rounds of workload submissions. Each round has 50 000 CPU-intensive, memory-intensive, and IO-intensive workloads, respectively, whose parameters are set the same as in the previous experiments. We have 10 cloud providers, and each of them has three cloud servers with the same initial state, i.e., server 1: (80, 50, 50), server 2: (50, 80, 50), and server 3: (50, 50, 80). The probability of obtaining a workload is proportional to $score_{\text{Total}}$. The results are shown in Fig.10. In this figure, the bars represent the total number of workloads achieved by cloud providers, and the curves represent the normal rate of each cloud provider's workloads.

Fig.10(a) shows the results of the case that CP_1–CP_3, CP_4–CP_6, and CP_7–CP_9 have 5, 3, and 2 rounds of SV degradation, respectively, during which 90% of their workloads experience 90% performance degradation. We can find that the cloud providers with higher performance stability (normal rate) can get more workloads, which can draw the same conclusion in Fig.9(a). Different from Fig.10(a), Fig.10(b) shows the results of the case that CP_1–CP_3 present their real performance to avoid four rounds of SV degradation, but reduce all the resource scores by 10. Comparing Fig.10(a) and Fig.10(b), we also have the same conclusion in Fig.9(b). Under AoCP's incentive of maintaining SV, the workload execution time in total can be reduced by 22.0% on average if all the cloud providers maintain their $score_{SV}$

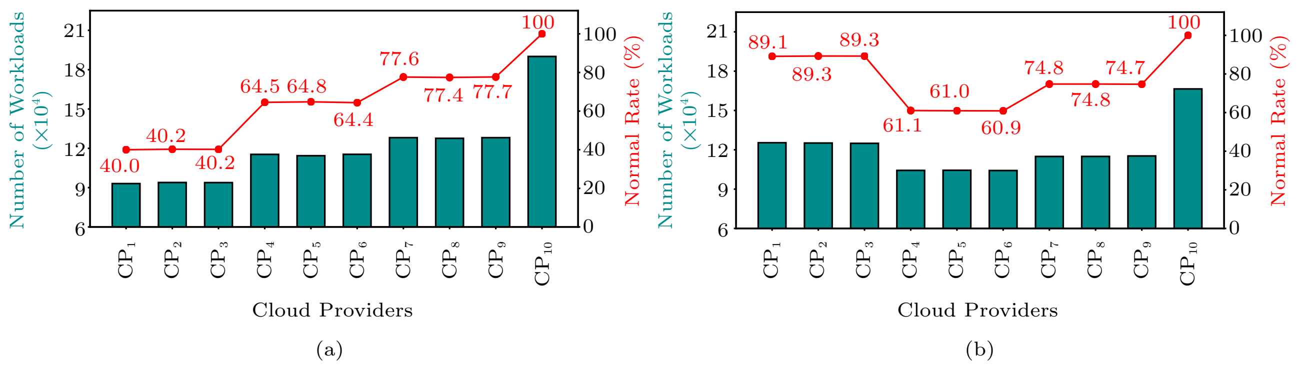


Fig.10. Statistical results of large-scale simulation experiments. (a) The case that some cloud providers cannot provide the performance as they claimed. (b) The case that CP_1–CP_3 present their real performance to avoid the *SV* degradation.

at 100 all the time compared with Fig. 10(a). Under AoCP's incentive of presenting real performance, the workload execution time is reduced by 7.4% on average in Fig.10(b) compared with Fig.10(a).

6 Discussion

6.1 Generality of RODE

RODE uses two common characteristics of Intel SGX to implement the performance monitoring mechanism, i.e., secured memory protection and remote attestation, which also exist in most of the other TEEs on various CPU processor architectures, e.g., Intel TDX [20], AMD SEV-SNP [21], and ARM CCA [22]. Therefore, it is feasible to adapt RODE to other TEEs while maintaining the original functionalities, and RODE only needs the cloud providers to be TEE-enabled. Although the TEE technique has not been fully adapted to all types of CPU processors, most processor manufacturers are exploring it to deal with various security issues, especially for the scenario of executing workloads requiring security guarantees in untrusted remote servers [17, 20–22]. Some decentralized cloud platforms (e.g., iExec and Golem) are also attempting to integrate TEE techniques to protect the integrity of the users' programs and guarantee the execution results. In addition, the development of TEE virtualization technologies [37, 38] can make cloud users use the TEE technique on the cloud servers without root access, and we also use the SGX's docker container for the evaluation in Section 5. Since Intel TDX is under development and we do not have servers with other CPU architectures, we introduce the design and implement RODE with Intel SGX to validate the effectiveness.

RODE provides a reliable assessment of cloud providers' performance to meet the needs of users, who attach importance to performance and cannot trust cloud providers. In addition, cloud providers also need a transparent and reliable performance assessment solution to demonstrate their high performance. In the face of these demands, cloud providers and cloud users have the motivation to utilize RODE, and pay for the performance assessment cost together, i.e., cloud providers reduce the cloud service prices a little and cloud users undertake a small reduction in the workload execution performance. To sum up, RODE is of generality to a large extent and is expected to become the key technique to enable robust performance assessment of decentralized cloud computing.

6.2 Performance Assessment Metrics

As we focus on the batch workloads with the IaaS mode, RODE mainly considers cloud providers' computation performance as the performance assessment metric for the workload execution. Moreover, the metric of computation performance can be forged by cloud providers, and thus RODE conducts reliability design for it. Other performance assessment metrics, e.g., network delay and consistency delay, can also affect the performance of the decentralized clouds, but are not the focus of this paper. In addition, these metrics can be directly obtained and reliably recorded without additional reliability design.

7 Conclusions

We proposed RODE, a robust solution to enable performance assessment for cloud providers of the decentralized cloud in an environment of mutual distrust. Empowered by the TEE technique, RODE's performance monitoring mechanism can generate reliable and verifiable performance results for workloads executed on untrusted cloud providers. Based on the reliable performance monitoring results, RODE's AoCP mechanism can reliably assess the performance of cloud providers in the decentralized cloud and incentivize cloud providers to offer real and stable performance for users' workloads. Our experiments showed that the performance monitoring mechanism of RODE can decrease the performance monitoring error rate by 77.5% and decrease the monitoring overhead by 98.4% compared with the state-of-the-art work. Besides, via extensive prototype, emulation, and simulation-based experiment results, the feasibility, effectiveness, and efficiency of RODE's AoCP mechanism are evaluated by the fact that the mechanism indeed incentivizes the cloud providers to offer performance the same as they claimed. Since RODE can be adapted to most of the TEEs while maintaining the original functionalities, exploring other TEE techniques except that we used for RODE may have the potential to further increase the performance monitoring accuracy and reduce monitoring overhead, which will be interesting future work.

References

[1] Liu H. A measurement study of server utilization in public clouds. In *Proc. the 9th International Conference on Dependable, Autonomic and Secure Computing*, December 2011, pp.435-442. DOI: 10.1109/DASC.2011.87.

[2] Zheng Z, Xie S, Dai H N, Chen X, Wang H. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 2018, 14(4): 352-375. DOI: 10.1504/IJWGS.2018.10016848.

[3] Dejun J, Pierre G, Chi C H. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proc. the 7th International Conference on Service Oriented Computing Workshop*, November 2009, pp.197-207. DOI: 10.1007/978-3-642-16132-2_19.

[4] Iosup A, Yigitbasi N, Epema D. On the performance variability of production cloud services. In *Proc. the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2011, pp.104-113. DOI: 10.1109/CCGrid.2011.22.

[5] Garfinkel S. An evaluation of Amazon's grid computing services: EC2, S3, and SQS. Technical Report, Harvard University, 2007. https://dash.harvard.edu/bitstream/handle/1/24829568/tr-08-07.pdf?sequence=1&isAllowed=y, May 2022.

[6] Leitner P, Cito J. Patterns in the chaos—A study of performance variation and predictability in public IaaS clouds. *ACM Transactions on Internet Technology*, 2016, 16(3): Article No. 15. DOI: 10.1145/2885497.

[7] Cai X, Shi J, Yuan R, Liu C, Zhen W, Chen Q, Li C, Leng J, Guo M. OVERSEE: Outsourcing verification to enable resource sharing in edge environment. In *Proc. the 49th International Conference on Parallel Processing*, August 2020, Article No. 71. DOI: 10.1145/3404397.3404409.

[8] Uriarte R B, DeNicola R. Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards. *IEEE Communications Standards Magazine*, 2018, 2(3): 22-28. DOI: 10.1109/MCOMSTD.2018.1800020.

[9] Aslani R, Hakami V, Dehghan M. A token-based incentive mechanism for video streaming applications in peer-to-peer networks. *Multimedia Tools and Applications*, 2018, 77(12): 14625-14653. DOI: 10.1007/s11042-017-5051-9.

[10] Samuel MD, Balakrishnan R. A grade-based incentive mechanism with starvation prevention for maintaining fairness in peer-to-peer networks. *International Journal of Systems Assurance Engineering and Management*, 2012, 3(2): 84-99. DOI: 10.1007/s13198-012-0098-5.

[11] Zeng F, Chen Y, Yao L, Wu J. A novel reputation incentive mechanism and game theory analysis for service caching in software-defined vehicle edge computing. *Peer-to-Peer Networking and Applications*, 2021, 14(2): 467-481. DOI: 10.1007/s12083-020-00985-4.

[12] Kang J, Xiong Z, Niyato D, Xie S, Zhang J. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal*, 2019, 6(6): 10700-10714. DOI: 10.1109/JIOT.2019.2940820.

[13] Nwebonyi F N, Martins R, Correia M E. Reputation based approach for improved fairness and robustness in P2P protocols. *Peer-to-Peer Networking and Applications*, 2019, 12(4): 951-968. DOI: 10.1007/s12083-018-0701-x.

[14] Paudel A, Chaudhari K, Long C, Gooi H B. Peer-to-peer energy trading in a prosumer-based community microgrid: A game-theoretic model. *IEEE Transactions on Industrial Electronics*, 2018, 66(8): 6087-6097. DOI: 10.1109/TIE.2018.2874578.

[15] Tushar W, Yuen C, Mohsenian-Rad H, Saha T, Poor H V, Wood K L. Transforming energy networks via peer-to-peer energy trading: The potential of game-theoretic approaches. *IEEE Signal Processing Magazine*, 2018, 35(4): 90-111. DOI: 10.1109/MSP.2018.2818327.

[16] Chen K, Lin J, Song Y. Trading strategy optimization for a prosumer in continuous double auction-based peer-to-peer market: A prediction-integration model. *Applied Energy*, 2019, 242: 1121-1133. DOI: 10.1016/j.apenergy.2019.03.094.

[17] Costan V, Devadas S. Intel SGX explained. https://eprint.iacr.org/2016/086.pdf, May 2022.

[18] Hoekstra M, Lal R, Pappachan P, Phegade V, Del Cuvillo J. Using innovative instructions to create trustworthy software solutions. In *Proc. the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, June 2013, Article No. 11. DOI: 10.1145/2487726.2488370.

[19] Sabt M, Achemlal M, Bouabdallah A. Trusted execution environment: What it is, and what it is not. In *Proc. the 2015 IEEE Trustcom/BigDataSE/ISPA*, August 2015, pp.57-64. DOI: 10.1109/Trustcom.2015.357.

[20] Sahita R, Caspi D, Huntley B, Scarlata V, Chaikin B, Chhabra S, Aharon A, Ouziel I. Security analysis of confidential-compute instruction set architecture for virtualized workloads. In *Proc. the 2021 International Symposium on Secure and Private Execution Environment Design*, September 2021, pp.121-131. DOI: 10.1109/SEED51797.2021.00024.

[21] AMD. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. White Paper, 2020, https://www.amd.com/system/_les/TechDocs/SEV-SNP-s trengthening-vm-isolation-with-integrity-protection-and-more.pdf, May 2022.

[22] Mulligan D P, Petri G, Spinale N, Stockwell G, Vincent H J. Confidential computing—A brave new world. In *Proc. the 2021 International Symposium on Secure and Private Execution Environment Design*, September 2021, pp.132-138. DOI: 10.1109/SEED51797.2021.00025.

[23] Naseri A, Navimipour N J. A new agent-based method for QoS-aware cloud service composition using particle swarm optimization algorithm. *Journal of Ambient Intelligence and Humanized Computing*, 2019, 10(5): 1851-1864. DOI: 10.1007/s12652-018-0773-8.

[24] Jatoth C, Gangadharan G R, Fiore U, Buyya R. SELCLOUD: A hybrid multi-criteria decision-making model for selection of cloud services. *Soft Computing*, 2019, 23(13): 4701-4715. DOI: 10.1007/s00500-018-3120-2.

[25] Temglit N, Chibani A, Djouani K, Nacer M A. A distributed agent-based approach for optimal QoS selection in web of object choreography. *IEEE Systems Journal*, 2017, 12(2): 1655-1666. DOI: 10.1109/JSYST.2016.2647281.

[26] Ding S, Wang Z, Wu D S, Olson D L. Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. *Decision Support Systems*, 2017, 93: 1-10. DOI: 10.1016/j.dss.2016.09.001.

[27] Al-Faifi A M, Song B, Hassan M M, Alamri A, Gumaei A. Performance prediction model for cloud service selection from smart data. *Future Generation Computer Systems*, 2018, 85: 97-106. DOI: 10.1016/j.future.2018.03.015.

[28] Xia Y, Zhou M C, Luo X, Zhu Q, Li J, Huang Y. Stochastic modeling and quality evaluation of infrastructure-as-a-service clouds. *IEEE Transactions on Automation Science and Engineering*, 2013, 12(1): 162-170. DOI: 10.1109/TASE.2013.2276477.

[29] Li L, Liu M, Shen W, Cheng G. Recommending mobile services with trustworthy QoS and dynamic user preferences via FAHP and ordinal utility function. *IEEE Transactions on Mobile Computing*, 2020, 19(2): 419-431. DOI: 10.1109/TMC.2019.2896239.

[30] Ardagna D, Barbierato E, Evangelinou A, Gianniti E, Gribaudo M, Pinto T, Guimaraes A, Silva A, Almeida J. Performance prediction of cloud-based big data applications. In *Proc. the 2018 ACM/SPEC International Conference on Performance Engineering*, April 2018, pp.192-199. DOI: 10.1145/3184407.3184420.

[31] Zou W, Lo D, Kochhar P S, Le X D, Xia X, Feng Y, Chen Z, Xu B. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 2019, 47(10): 2084-2106. DOI: 10.1109/TSE.2019.2942301.

[32] Wang P, Meng J, Chen J, Liu T, Zhan Y, Tsai W, Jin Z. Smart contract-based negotiation for adaptive QoS-aware service composition. *IEEE Transactions on Parallel and Distributed Systems*, 2018, 30(6): 1403-1420. DOI: 10.1109/TPDS.2018.2885746.

[33] Viriyasitavat W, Da Xu L, Bi Z, Hoonsopon D, Charoenruk N. Managing QoS of internet-of-things services using blockchain. *IEEE Transactions on Computational Social Systems*, 2019, 6(6): 1357-1368. DOI: 10.1109/TCSS.2019.2919667.

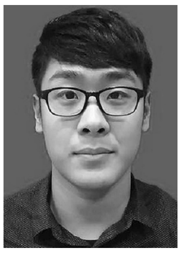
[34] Mauthe N, Kargen U, Shahmehri N. A Large-Scale empirical study of Android app decompilation. In *Proc. the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering*, March 2021, pp.400-410. DOI: 10.1109/SANER50967.2021.00044.

[35] Pawlowski A, Contag M, Van Der Veen V, Ouwehand C, Holz T, Bos H, Athanasopoulos E, Giuffrida C. MARX: Uncovering class hierarchies in C++ programs. In *Proc. the 24th Annual Network and Distributed System Security Symposium*, Feb. 26-Mar. 1, 2017. DOI: 10.14722/ndss.2017.23096.

[36] Katz D S, Ruchti J, Schulte E. Using recurrent neural networks for decompilation. In *Proc. the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering*, March 2018, pp.346-356. DOI: 10.1109/SANER.2018.8330222.

[37] Arnautov S, Trach B, Gregor F *et al.* SCONE: Secure Linux containers with Intel SGX. In *Proc. the 12th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.689-703.

[38] Tsai C C, Porter D E, Vij M. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *Proc. the 2017 USENIX Annual Technical Conference*, July 2017, pp.645-658.



Jiu-Chen Shi received his B.S. degree in software engineering from Dalian University of Technology, Dalian, in 2019. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering of Shanghai Jiao Tong University, Shanghai. His research interests include resource management in various architectures, high-performance computing, and trusted computing.

Xiao-Qing Cai received her B.S. degree in computer science from Northeastern University, Shenyang, in 2017. She is currently a Ph.D. candidate in the Department of Computer Science and Engineering of Shanghai Jiao Tong University, Shanghai. Her research interests include blockchain, trusted computing, and resource management.

Wen-Li Zheng received his Ph.D. degree in electrical and computer engineering from Ohio State University, Ohio, in 2016. He is currently a tenure-track associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include energy-efficient computing, big data, distributed machine learning, blockchain, and trusted computing.

Quan Chen received his Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 2014. He is currently a professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include high-performance computing, task scheduling in various architectures, resource management in datacenter, runtime system, and operating system.

De-Ze Zeng received his Ph.D. and M.S. degrees in computer science from University of Aizu, Aizu-Wakamatsu, in 2013 and 2009, respectively. He is currently a full professor in School of Computer Science, China University of Geosciences, Wuhan. His current research interests include edge computing, cloud computing, and future networking technologies. He has authored three books and published over 100 papers in refereed journals and conferences in these areas. He serves in editorial boards of Journal of Network and Computer Applications, Frontiers of Computer Science, and Open Journal of the Computer Society. He is a senior member of CCF and a member of IEEE.

Tatsuhiro Tsuchiya received his M.E. and Ph.D. degrees in engineering from Osaka University, Osaka, in 1995 and 1998, respectively. He is currently a professor of the Graduate School of Information Science and Technology at Osaka University, Osaka. His research interests are in the areas of model checking, software testing, and distributed fault-tolerant systems.

Min-Yi Guo received his Ph.D. degree in computer science from the University of Tsukuba, Tsukuba, in 1998. He is currently a Zhiyuan Chair Professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. His present research interests include parallel/distributed computing, compiler optimizations, embedded systems, big data, and cloud computing. He is now on the editorial board of IEEE Transactions on Parallel and Distributed Systems and Journal of Parallel and Distributed Computing. Dr. Guo is an IEEE Fellow and a CCF Fellow.