Received August 4, 2020, accepted August 19, 2020, date of publication September 1, 2020, date of current version September 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020863

Shared-Mode Resource Allocation for Cloud-Based Load Testing

WENMING JIN, JU QIAN, (Member, IEEE), AND SHUOYAN YAN
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
MIIT Key Laboratory of Safety-Critical Software, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China

Corresponding author: Ju Qian (jqian@nuaa.edu.cn)

This work was supported by the Equipment Pre-Research Project of China under Grant 41402020501-170441402030.

ABSTRACT Resource allocation is essential for cloud-based load testing. The existing techniques use coarse-grained resource allocation methods with an entire virtual machine occupied by a single test task for cloud-based load testing. The idle resources in a virtual machine are unable to be used by other load testing tasks. This may result in uneconomical use of test resources and increase test costs. To optimize the use of test resources, this paper presents a shared-mode resource allocation method for cloud-based load testing. The method shares client-side virtual machine resources among load testing tasks. It takes minimizing resource redundancy, test execution cost, and network communication cost as optimization objectives of resource allocation, with the assurance of enough test resources as a basic constraint. We introduce a multi-objective optimization algorithm to create an optimized resource allocation plan for load testing tasks within a time window. The experiments show that the proposed method can reduce resource demands for load testing and thereby save the test costs.

INDEX TERMS Cloud testing, load testing, test resource allocation, multi-objective optimization.

I. INTRODUCTION

Many failures in online services are due to their inability to scale to meet user demands [1]. To ensure service quality, it is often necessary to conduct load testing before product releases [2], [3]. Load testing is usually performed by simulating workloads from a cluster of client hosts to test the responses of a service [1]. To enable large-scale load testing, various issues need to be addressed, such as the construction and maintenance of the client hosts, the installation of the simulation agents, etc. These issues are costly to be addressed in a local manner [4]. Nowadays, load testing is often migrated to the cloud and conducted based on cloud resources. Performing load testing in the cloud can ease the setup of test environments and reduce the hardware purchase and maintenance cost [5]–[7].

To migrate load testing to the cloud, certain resource allocation and scheduling techniques need to be introduced [8]. The resource allocation that allocates client-side virtual machines for the simulation of workloads is essential to the effective execution of load testing tasks and the operation costs of cloud testing providers [9]. In the existing work, the techniques in [10]–[12] allocate resources for test tasks using an exclusive utilization mode of virtual machine resources, with one virtual machine occupied by at most a single test task. Commercial cloud testing services like Tencent WeTest [13] and Alibaba PTS [14] also adopt exclusive-mode virtual machine resource allocation in their test-script-driven load testing. Under the exclusive mode, the virtual machines in the cloud are used in a coarse-grained manner, and a virtual machine can simultaneously provide test services for only one single load testing task. With this granularity, once a virtual machine is assigned to a test task, no other load testing task can use the idle resources in the virtual machine until the assigned task finishes its execution. This may easily result in inefficient use of the test resources and increase the test costs.

Shared-mode resource allocation can improve resource utilization efficiency and minimize resource waste [15]. Many cloud systems [16]–[18] share processors, clusters, and virtual machine resources among their cloud tasks. In cloud-based load testing, shared-mode resource allocation suggests that a virtual machine can be allocated to multiple load testing tasks at the same time. A test task no longer occupies entire virtual machine resources, and the unit of resource allocation

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Abdur Razzaque.

is refined from a whole virtual machine to parts of its computing resources. With shared-mode resource allocation, it is possible to use limited virtual machine resources to run more load testing tasks simultaneously.

However, there is a lack of shared-mode resource allocation methods in the literature for cloud-based load testing. The existing shared-mode resource allocation techniques for common cloud tasks cannot straightforwardly be extended to cloud-based load testing. The reasons are two folds. First, load testing exhibits a complex many-to-many task-resource model, where a load testing task may require multiple virtual machines for execution, and a virtual machine can run multiple load testing tasks at the same time. This is different from the common cloud task scheduling, where a task is only scheduled onto one virtual resource [19]. Second, the resource allocation constraints and optimization objectives of load testing tasks are also different from common cloud tasks. Load testing uses virtual machine resources in a cooperative way for test execution. We want the resource allocation to ensure the quality of service and minimize the amount and costs of the used resources. Although such goals are similar to many ones in other cloud resource allocation and scheduling [19], [20], how to express and achieve these task-specific goals is more complex than that using a single virtual machine resource to accomplish a cloud task.

To fill the gap, we propose a shared-mode resource allocation method for cloud-based load testing in this paper. The method takes minimizing resource redundancy, test execution cost, and network communication cost as the optimization objectives of resource allocation. It also regards ensuring enough virtual machine resources for test tasks as a basic constraint. With these objectives and constraints, the amount and costs of the used resources can possibly be reduced, and the load testing tasks can be effectively executed. Since the optimization objectives are not always consistent, we design a multi-objective optimization algorithm for resource allocation. The algorithm allocates shared virtual machine resources for load testing tasks within a time window. The whole approach can optimize the resource utilization of load testing, and thereby reduce test costs. Our experimental results show that for the tested cloud environments and load testing tasks, the shared-mode resource allocation method performs better than the exclusive-mode one in terms of the resource utilization efficiency. Compared with the exclusive-mode resource allocation, the shared-mode resource allocation reduced the resource redundancy by more than 12.4%, the test execution cost by over 8.6%, and the numbers of occupied virtual machines and physical machines by more than 15.9% and 10.2%, respectively.

The remainder of the paper is organized as follows. Section II highlights the related work. Section III introduces some backgrounds for cloud-based load testing. We present the basic model of our shared-mode resource allocation in Section IV. Section V introduces the multi-objective shared-mode resource allocation algorithm built on genetic evolution. Section VI shows the experimental results. Finally, we conclude the paper in Section VII.

II. RELATED WORK

Resource allocation for cloud tasks is, in general, an NP-hard problem [19]. In this area, Keshanchi *et al.* [21] proposed an improved genetic algorithm to allocate processor resources for cloud tasks. Sun *et al.* [22] introduced a QoS-oriented modeling framework to allocate optimized cloud servers for web applications to meet the web applications' QoS goals. Aladwani [23] introduced a method named TC&VC to classify cloud tasks based on task lengths and then allocate virtual machines for the tasks according to the classification. Zhang and Zhou [24] proposed a two-stage task scheduling framework to allocate virtual machines for cloud tasks. In the first stage, they mark tasks with the attributes of the demanded virtual machines. In the second stage, they assign suitable virtual machines to these tasks with the objective of minimizing unreasonable resource allocation. Arunarani *et al.* [20] presented a comprehensive survey of task resource allocation strategies and the corresponding metrics suitable for cloud computing environments. These resource allocation methods for general cloud tasks provide references for solving many problems, but they are not directly applicable to allocating test resources for cloud-based load testing tasks.

For the allocation of test resources in cloud environments, Kang *et al.* [10] proposed a resource allocation method based on improved Particle Swarm Optimization (PSO) to allocate virtual machines for test tasks, which can improve the efficiency of cloud resource allocation. Lampe *et al.* [11] presented a model for scheduling software tests on a Testing-as-a-Service system. Based on the model, they analyzed the resource utilization under a set of scheduling algorithms, e.g., Smallest Job Longest Operation First, Shortest Operation First, and Longest Operation First. Lu *et al.* [12] attempted to solve the automatic test task scheduling problem (TTSP) with the objectives of minimizing the maximal test completion time and the mean workloads of virtual machines. A formal model of the TTSP is established, and a chaotic non-dominated sorting genetic algorithm is presented to solve the problem. For load testing, [6], [9] use techniques like admission control to allocate virtual machine resources for sending client requests, but they do not use virtual machines in a shared mode. In the above methods, one virtual machine can process at most a single test task at a time. As discussed in Section I, such exclusive-mode resource allocation may not efficiently use the test resources.

Some existing work shares processor, cluster, and virtual machine resources between cloud tasks. In multiprocessor scheduling, Agrawal and Baruah [16] proposed a measurement-based model for resource allocation of parallel real-time tasks. It allows the idle processor resources in a virtual machine to be used by more than one real-time task. Cano *et al.* [17] introduced a framework sharing cloud cluster (virtual machine cluster) resources for web service and

Map/Reduce applications. On the framework, the resource optimization problem is formulated as a non-linear mathematical programming model to increase cluster utilization. Zhu and Du [18] divided cloud tasks into multiple sub-tasks based on the Map/Reduce programming model. These sub-tasks can execute parallelly and share server computing resources. The above methods all employ shared-mode resource allocation. The resource sharing implies that idle resources in a virtual machine/cluster executing certain tasks can be allocated to other tasks, and multiple tasks can execute in parallel on the same virtual machine/cluster by sharing resources. Compared with the tasks supported by these methods, cloud-based load testing tasks are more complex than general cloud tasks in terms of the execution model and the resource optimization objectives and constraints (Section I). The existing shared-mode resource allocation methods cannot be used for cloud-based load testing. Therefore, we design a new resource allocation method in this work.

III. BASIC CONCEPTS IN CLOUD-BASED LOAD TESTING

We deploy workload generators to the cloud and use them to create client-side access pressures for cloud-based load testing (Fig.1). When the target load scale is large, the workloads need to be generated from multiple client-side virtual machines. In the traditional style, one virtual machine is used to generate workloads for a single load testing task; while in shared-mode resource allocation, a virtual machine can be used to generate workloads for multiple load testing tasks.

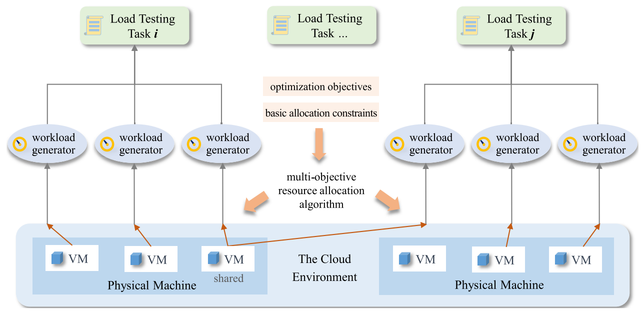


FIGURE 1. The cloud-based load testing.

1) CLOUD-BASED LOAD TESTING TASK

A cloud-based load testing task is a task to create and execute workloads from client-side virtual machines deployed in the cloud to evaluate the performance of a service under test. The workloads are often expressed via test scripts, which can be a JMeter [25] test script, a Selenium [26] test script, a FunkLoad-like program [27], etc. describing the behavior of a client.

More formally, a cloud-based load testing task T can be modeled as a tuple $T = \langle SUT, Script, Load_{max}, duration\rangle$, where:

- SUT is the service under test;
- $Script = [s_1, s_2, \ldots, s_n]$ represents a vector of test scripts expressing the workloads in the load testing task;
- $Load_{max} = [maxload(s_1), \ldots, maxload(s_n)]$ represents the maximum load scale for each test script, where $maxload(s_i)$ is the maximum load for the script s_i;
- $duration$ is the execution time duration of the task.

The resources required to parallelly run a test script s at a load scale $load$ can be estimated by a function $Est : s \times load \to R$. The function can be manually provided or learned from the historical records of running a test script at small scales before doing load testing. For a load testing task T, the required resources can be estimated as following,

$$R(T) = \sum_{s \in Script(T)} Est(s, maxload(s)) = [cpu, ram, bw],$$

where cpu, ram, and bw represent the CPU (times of the base frequency $\times$ total CPU utilization of multiple cores), memory, and network bandwidth resources required by the load testing task T.

Since the maximum load scale $Load_{max}$ of a load testing task is often large, a lot of resources may be required to execute the task, and these resources usually are difficult to be provided by a single virtual machine.

2) VIRTUAL MACHINES

We assign virtual machines for different load testing tasks in order to simulate the workloads. The total set of virtual machines in the cloud testing environment can be represented as $VM_{global} = \{vm_1, vm_2, \ldots, vm_n\}$. Each virtual machine vm in VM_{global} can be modeled as a tuple $vm = \langle pid, R_v, Tasks, state\rangle$, where:

- pid is the identifier of the physical machine where the virtual machine belongs to;
- $R_v = [cpu_v, ram_v, bw_v]$ represents the available CPU, memory, and network bandwidth resources of the virtual machine, respectively;
- $Tasks$ is the set of load testing tasks being executed on the virtual machine;
- $state \in \{off, on\}$, denoting the virtual machine is either shut down or in a running state, respectively.

The objective of our resource allocation is to determine the virtual machines bound to each test task or, from another perspective, the set $Tasks$ for each virtual machine. In the shared-mode resource allocation, a set $Tasks$ may contain more than one element. When there is no task assigned to a virtual machine, the virtual machine can be shut down.

3) CLOUD TESTING ENVIRONMENT

The structure of the cloud testing environment affects the assignment of virtual machines. We use an undirected graph in Fig.2 to demonstrate the cloud testing environment. A cloud testing environment G can be modeled as a tuple $G = \langle VM_{global}, PM_{global}, R_{global}, E\rangle$, where:

- VM_{global} represents the set of virtual machines in G, e.g., $\{vm_1, \ldots, vm_6\}$ in Fig.2;
- PM_{global} represents the set of physical machines in G, e.g., $\{pm_1, pm_2\}$ in Fig.2;

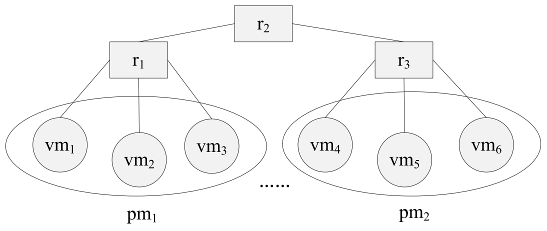


FIGURE 2. A cloud testing environment.

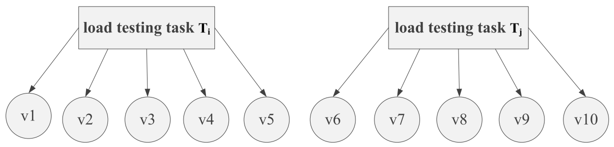


FIGURE 3. Demonstration of the exclusive-mode resource allocation.

- R_{global} represents the set of routers in the cloud testing environment, e.g., $\{r_1, r_2, r_3\}$ in Fig.2;
- E is an edge set, where an edge connecting two nodes (virtual machines or routers) represents the communication link between them.

IV. SHARED-MODE RESOURCE ALLOCATION MODEL

We present the basic model of the shared-mode test resource allocation in this section. The below will introduce the resource allocation plan used in our cloud-based load testing, the optimization objectives of the resource allocation, and the constraints on the resource allocation. The detailed algorithm used to determine the resources allocated for different load testing tasks is designed based on the model and will be presented in the next section.

A. SHARED-MODE RESOURCE ALLOCATION PLAN

Given a sequence of load testing tasks $TS = \langle T_1, T_2, \ldots, T_m \rangle$, we call the assignments of virtual machines to the load testing tasks in TS a resource allocation plan for TS. A resource allocation plan can be denoted as a vector P, each row of which is a set of virtual machines to be allocated for a corresponding load testing task in the task sequence TS:

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \ldots \\ P_m \end{bmatrix} = \begin{bmatrix} \{vm_{11}, vm_{12}, \ldots, vm_{1c_1}\} \\ \{vm_{21}, vm_{22}, \ldots, vm_{2c_2}\} \\ \ldots. \\ \{vm_{m1}, vm_{m2}, \ldots, vm_{mc_m}\} \end{bmatrix}.$$

We use a map F from a load testing task sequence TS to a resource allocation plan P to represent the generation of resource allocation plans for load testing,

$$F : TS \rightarrow P.$$

The existing exclusive-mode test resource allocation methods take a virtual machine as an allocation unit, a demonstration of which is shown in Fig. 3. In the figure, we allocate a set of virtual machines $VM_1 = \{v1, v2, v3, v4, v5\}$ for the load testing task T_i. The idle resources in these virtual machines cannot be used by another load testing task T_j. We need to allocate another set of virtual machines $VM_2 = \{v6, v7, v8, v9, v10\}$ for the load testing task T_j. This may lead to wastes of test resources.

To reduce resource wastes, this paper proposes a shared-mode resource allocation method, in which a load testing task no longer occupies entire virtual machine resources to generate workloads. This allows a virtual machine to execute multiple load testing tasks at the same time. A demonstration of such resource allocation is shown in Fig. 4. In the figure, we allocate a set of virtual machines $VM_1 = \{v1, v2, v3, v4, v5\}$ to the load testing task T_i and a set of virtual machines $VM_2 = \{v2, v4, v5, v6, v7, v8\}$ to the load testing task T_j. The virtual machines $v2$, $v4$, and $v5$ are shared by both T_i and T_j. Compared with the exclusive mode, the shared-mode resource allocation uses test resources in a more fine-grained way. It can reduce the number of the required virtual machines and thereby save the costs of load testing as much as possible.

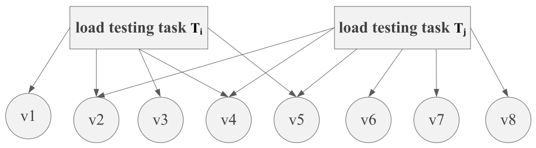


FIGURE 4. Demonstration of the shared-mode resource allocation.

B. OPTIMIZATION OBJECTIVES FOR RESOURCE ALLOCATION

An optimized test resource allocation needs to not only minimize the used test resources in order to reduce the test costs but also ensure the efficiency of testing execution. A resource allocation plan should meet multiple objectives as much as possible. These objectives may conflict with each other and can be coordinated by multi-objective optimization algorithms [28].

1) MINIMIZING RESOURCE REDUNDANCY

The total resources in the virtual machine set $VM(TS, P)$ allocated to a load testing task sequence TS under a resource allocation plan P might be more than the actual requirements of the tasks in TS. This leads to redundancy in the allocated test resources. To reduce redundancy and avoid wastes, this paper introduces an optimization objective of minimizing resource redundancy for the load testing task sequence TS. We define a resource redundancy function $L(TS, P)$ to denote the difference between the total available resources in the allocated virtual machines and the minimal resources required for load testing,

$$L(TS, P) = \omega_1 \times \Big(\sum_{vm_i \in VM(TS,P)} R(vm_i) - \sum_{T_i \in TS} R(T_i) \Big),$$

where $R(vm_i)$ represents the vector of available resources in a virtual machine vm_i, $R(T_i)$ represents the resource requirements of a load testing task T_i, and ω_1 is the resource weight vector. The smaller value of the function $L(TS, P)$, the fewer wasted resources.

2) MINIMIZING TEST EXECUTION COST

Cloud service providers charge money for the use of virtual machines according to the scale of the resources and the use duration, e.g., [29]. This brings test execution costs for cloud-based load testing. We also introduce an optimization objective to minimize such test execution costs. The work mainly considers the occupancy cost of virtual machines (cloud instances) during testing execution. Given a load testing task sequence TS and a resource allocation plan P for TS, let $VM(TS, P)$ be the set of virtual machines allocated for TS in plan P. We define a test execution cost function $Z(TS, P)$ to denote the occupancy cost of all the virtual machines in $VM(TS, P)$,

$$Z(TS, P) = \sum_{vm_i \in VM(TS,P)} price(vm_i) * occupyTime(vm_i).$$

$price(vm_i)$ denotes the per time unit occupancy cost of a virtual machine vm_i. The price depends on the hardware resources in vm_i, $price(vm_i) = \omega_2 * R(vm_i)$. ω_2 is a cost coefficient vector, and $R(vm_i)$ represents the total resources of vm_i. $occupyTime(vm_i)$ denotes the occupancy time of the virtual machine vm_i, which depends on the maximum execution duration of the test tasks allocated to vm_i in the load testing task sequence TS. The smaller value of the function $Z(TS, P)$, the lower test cost.

3) MINIMIZING NETWORK COMMUNICATION COST

When executing a load testing task, the requests from the clients are routed to the service under test (SUT). The more complex routing paths, the higher cost of network communication. This may result in inefficient execution of load testing and lead to longer average response time of workloads which does not reflect the actual performance of the SUT. To ensure the efficiency of load testing execution, we introduce an optimization objective of minimizing network communication cost for a load testing task sequence.

For easy estimation of the network communication cost, we assume the requests passing through each router were with the same time delay. The minimum number of routers that the requests go through between two virtual machines vm_i and vm_j, denoted as $link(vm_i, vm_j)$, is used to determine the communication cost between these two virtual machines. For example, for a network shown in Fig. 5, $link(vm_1, vm_s) = link(vm_2, vm_s) = 2$, $link(vm_3, vm_s) = link(vm_4, vm_s) = link(vm_5, vm_s) = 1$.

Let $VM(T_i, P)$ be the set of virtual machines allocated for a load testing task T_i in a task sequence TS under a resource allocation plan P, and assume the SUT is deployed in node $SUT(T_i)$. We define a network communication cost function $N(TS, P)$ to estimate the network communication cost of a load testing task sequence TS under a resource

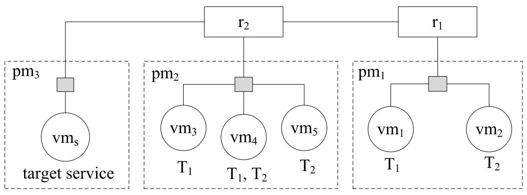


FIGURE 5. An example network structure.

allocation plan P,

$$N(TS, P) = \sum_{T_i \in TS} \frac{delay \times \sum_{vm \in VM(T_i,P)} link(vm, SUT(T_i))}{|VM(T_i, P)|}.$$

$N(TS, P)$ is the sum of the average network transmission delay of the virtual machine set corresponding to each load testing task in TS. $|VM(T_i, P)|$ is the number of virtual machines in $VM(T_i, P)$, and *delay* represents the time delay of the requests passing through a single router.

C. CONSTRAINT FOR RESOURCE ALLOCATION

To ensure that it is possible to generate the designated scale of access pressures for the service under test and the load testing tasks sharing resources can execute independently, we take the resource guarantee ability as a basic constraint. Let $VM(T_i, P)$ be the set of virtual machines allocated to a load testing task T_i ($T_i \in TS$) under a resource allocation plan P. The constraint requires that the total resources of the virtual machine set $VM(T_i, S)$ not being less than the minimum resources required to initiate the load testing task T_i, i.e.,

$$\sum_{vm \in VM(T_i,P)} R_{T_i}(vm) \geq R(T_i), \quad T_i \in TS.$$

$R_{T_i}(vm)$ represents the available resources allocated to the load testing task T_i in a virtual machine vm. $R(T_i)$ represents the resource requirements of the load testing task T_i.

V. MULTI-OBJECTIVE RESOURCE ALLOCATION ALGORITHM

In this work, our cloud-based load testing system allows users to submit load testing tasks at any time. All the submitted tasks form a time-labeled task sequence $TS = \langle t_1 : T_1, t_2 : T_2, \ldots, t_m : T_m \rangle$. In a large-scale cloud testing environment, the test center may receive a large number of load testing tasks when there are lots of users. If we process test tasks in a one-by-one manner, because the future arrivals of test tasks are not considered, the resource allocation plan may not be sufficiently optimized from an overall perspective. If we process test tasks in an offline batched mode, the efficient use of resources can be guaranteed, but the late responses of test requests may lead to bad user experience. As a compromise, this work enables nearly real-time resource allocation for load testing tasks based on a sliding window mechanism. A demonstration of the sliding window is shown in Fig. 6, where t_i is the arrival time of a load testing task T_i,

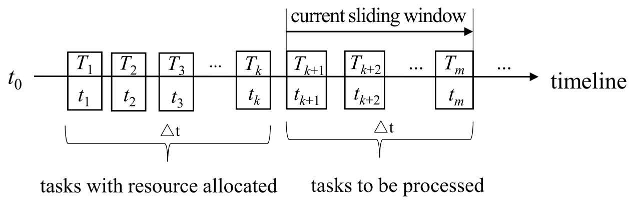


FIGURE 6. A demonstration of the sliding window.

and for two adjacent load testing tasks T_k and T_{k+1} on the timeline, $t_k \leq t_{k+1}$. Within each processing cycle, the sliding window moves to the right along the timeline for every Δt time. We handle all the test tasks in a window in a batched mode. The order between the tasks in the time window is ignored. We find an optimized resource allocation plan for all these tasks at one time, allocate test resources according to the plan, and start the load testing tasks in the time window simultaneously.

In a cloud testing environment G, for a load testing task sequence TS in a sliding window, the shared-mode resource allocation of TS can be expressed as a multi-objective planning problem. The problem is close to the Quadratic Multiple Knapsack Problem (QMKP) [30], where a virtual machine is an item, and a load testing task is a knapsack. Given n items and m knapsacks, the objective is to maximize the total value under the constraint of each knapsack's capacity. However, the test resource allocation for load testing tasks is more complicated than QMKP: 1) a single objective becomes multiple objectives; 2) the same virtual machine (item) can be allocated to multiple load testing tasks (knapsacks), and the fewer virtual machines, the better.

To generate an optimized resource allocation plan for a load testing task sequence, we propose a multi-objective optimization algorithm (Algorithm 1) based on genetic evolution. The algorithm takes the resource redundancy, test execution cost, and network communication cost estimation functions as the fitness functions. It first generates a number of resource allocation plans for TS to construct an initial population and uses combinations of variable-length integer sets to encode these resource allocation plans. Then, a child population is created by crossover and mutation operations. We repair and optimize the allocation plans in the child population. Next, we merge the parent and child populations into a mixed population. All the plans in the mixed population are ranked and sorted to derive a new optimized population. The process is repeated until reaching the maximum evolution generation. Finally, the algorithm outputs an optimized resource allocation plan for TS. More details about the operations in the algorithm will be introduced in the following subsections.

Algorithm 1 Multi-Objective Shared-Mode Resource Allocation Algorithm for Cloud-Based Load Testing

```
Input: A cloud testing environment $G$ and a load testing task sequence $TS$
Output: An optimized resource allocation plan for $TS$
 1: Capture a snapshot of $G$, including the network topological structure, the information of the physical machines and virtual machines, etc.;
 2: Initialize the parameters: the maximum evolution generation $K$, the population size $N$, etc.;
 3: Generate resource allocation plans for $TS$ to build an initial population $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$;
 4: Encode the resource allocation plans in $\mathcal{P}$;
 5: Calculate the objective function values for each allocation plan in $\mathcal{P}$;
 6: for $i = 1 \rightarrow K$ do
 7:     Create a child population by crossover and mutation;
 8:     Repair and optimize the child population;
 9:     Merge the parent and child populations into a mixed one;
10:     Rank and sort all the allocation plans in the mixed population to generate an optimized population;
11: end for
12: return the first plan in the ranked and sorted population;
```

A. ENCODING

The QMKP-like problems are often encoded by binary matrix. Let X be a binary matrix of $n \times m$. If a virtual machine i is allocated to a load testing task k, then $x_{ik} = 1$; otherwise, $x_{ik} = 0$. The number of virtual machines in a cloud testing environment can be huge, which will lead to a too sparse matrix. To address the problem, we introduce a combination of variable-length integer sets to encode resource allocation plans. A virtual machine is encoded as an integer, a resource allocation plan of a single load testing task is encoded as a variable-length integer set, and the resource allocation plan of the whole load testing task sequence is encoded as a combination of variable-length integer sets. Fig. 7 shows an example of the encoding. Assume there are three load testing tasks A, B, and C in the task sequence, and the virtual machine sets $\{vm_1, vm_2, vm_4\}$, $\{vm_3, vm_4, vm_5\}$, and $\{vm_5, vm_6, vm_7\}$ are allocated to A, B, and C, respectively. P represents the encoded resource allocation plan for the task sequence.

$$P = \begin{bmatrix} \{vm_1, vm_2, vm_4\} \\ \{vm_3, vm_4, vm_5\} \\ \{vm_5, vm_6, vm_7\} \end{bmatrix} = \begin{bmatrix} \{1, 2, 4\} \\ \{3, 4, 5\} \\ \{5, 6, 7\} \end{bmatrix}$$

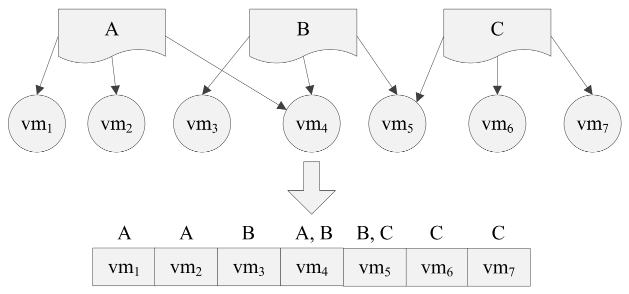


FIGURE 7. An example of the resource allocation plan encoding.

B. GENERATE CHILD POPULATION

As shown in step 7 of Algorithm 1, we take the last generation as the parent generation and create a child population of size N by crossover and mutation. The crossover operation ensures global searching ability and enables the algorithm to search for better resource allocation plans. The mutation operation adds diversity in the resource allocation plans and prevents premature convergence of the population.

We select two allocation plans f_1 and f_2 from the parent population with a crossover probability of p_c for crossover operation (example shown below). The crossover randomly picks a row in these plans as the intersection point, e.g., $\{vm_3, vm_4, vm_5\}$ in f_1, which corresponds to $\{vm_7, vm_8, vm_{13}\}$ in f_2. Then, it exchanges the rows in f_1 and f_2 after the selected row to construct two child allocation plans c_1 and c_2. The crossover process is essentially to replace allocation plans for test tasks.

$$\begin{bmatrix}\{vm_1, vm_2, vm_4\}\\ \{vm_3, vm_4, vm_5\}\\ \{vm_5, vm_6, vm_7\}\\ \{vm_8, vm_9\}\end{bmatrix}_{f_1} \otimes \begin{bmatrix}\{vm_6, vm_8, vm_{12}\}\\ \{vm_7, vm_8, vm_{13}\}\\ \{vm_{10}, vm_{12}\}\\ \{vm_{11}, vm_{13}, vm_{14}\}\end{bmatrix}_{f_2}$$
$$\rightarrow \begin{bmatrix}\{vm_1, vm_2, vm_4\}\\ \{vm_3, vm_4, vm_5\}\\ \{vm_{11}, vm_{13}, vm_{14}\}\\ \{vm_{10}, vm_{12}\}\end{bmatrix}_{c_1} \begin{bmatrix}\{vm_6, vm_8, vm_{12}\}\\ \{vm_7, vm_8, vm_{13}\}\\ \{vm_8, vm_9\}\\ \{vm_5, vm_6, vm_7\}\end{bmatrix}_{c_2}$$

We select an allocation plan P from the parent population with a mutation probability of p_m for the mutation operation. The mutation randomly picks a row (e.g., $\{vm_3, vm_4, vm_5\}$) in P, which is the resource allocation plan for a load testing task T_i, and reallocates a new enough-to-use virtual machine set (e.g., $\{vm_6, vm_8, vm_{10}\}$) for T_i to create a new resource allocation plan for the given task sequence.

$$P = \begin{bmatrix}\{vm_1, vm_2, vm_4\}\\ \{vm_3, vm_4, vm_5\}\\ \{vm_5, vm_6, vm_7\}\end{bmatrix} \rightarrow \begin{bmatrix}\{vm_1, vm_2, vm_4\}\\ \{vm_6, vm_8, vm_{10}\}\\ \{vm_5, vm_6, vm_7\}\end{bmatrix}$$

C. REPAIR AND OPTIMIZE CHILD POPULATION

As shown in step 8 of Algorithm 1, to ensure the validity and superiority of the resource allocation plans, we repair and optimize the child population created by the crossover and mutation.

After doing crossover and mutation, the child population may contain invalid resource allocation plans that violate the basic constraint (Section IV-C, the total resources should be enough for testing). This paper uses a repair algorithm (Algorithm 2) to make these plans satisfy the basic constraint. Take the following allocation plan P for example. Assume the total available resources of the virtual machines ($\{vm_5, vm_6\}$) allocated to the test task T_3 are less than the requirement. We randomly allocate a number of additional virtual machines with idle resources to T_3 for repair until P satisfies the basic constraint (steps 4-7 of Algorithm 2).

Algorithm 2 The Repair of a Resource Allocation Plan

```
Input: An invalid resource allocation plan $P$
Output: A valid resource allocation plan
1: for each load testing task $T_k$ in $P$ do
2:   let $VM_k$ be the virtual machine set allocated to $T_k$;
3:   calculate the total available resources in $VM_k$, $R_k = \sum_{vm \in VM_k} R_{T_k}(vm)$, where $R_{T_k}(vm)$ represents the available resources allocated to $T_k$ in $vm$;
4:   while $R_k < R(T_k)$ do
5:     randomly select a virtual machine $vm_i$, allocate it to load testing task $T_k$, and add $vm_i$ to $VM_k$;
6:     update $R_k$ and the available resources of $vm_i$;
7:   end while
8: end for
9: return $P$;
```

$$P = \begin{bmatrix}\{vm_1, vm_2, vm_4\}_{T_1}\\ \{vm_3, vm_4, vm_5\}_{T_2}\\ \{vm_5, vm_6\}_{T_3}\end{bmatrix}$$
$$\begin{cases} R_{T_1}(vm_1) + R_{T_1}(vm_2) + R_{T_1}(vm_4) > R(T_1)\\ R_{T_2}(vm_3) + R_{T_2}(vm_4) + R_{T_2}(vm_5) > R(T_2)\\ R_{T_3}(vm_5) + R_{T_3}(vm_6) < R(T_3)\end{cases}$$
$$\rightarrow \begin{bmatrix}\{vm_1, vm_2, vm_4\}_{T_1}\\ \{vm_3, vm_4, vm_5\}_{T_2}\\ \{vm_5, vm_6, vm_7\}_{T_3}\end{bmatrix}$$
$$\begin{cases} R_{T_1}(vm_1) + R_{T_1}(vm_2) + R_{T_1}(vm_4) > R(T_1)\\ R_{T_2}(vm_3) + R_{T_2}(vm_4) + R_{T_2}(vm_5) > R(T_2)\\ R_{T_3}(vm_5) + R_{T_3}(vm_6) + R_{T_3}(vm_7) > R(T_3)\end{cases}$$

Some valid resource allocation plans in the child population may allocate much more virtual machine resources than the minimum requirement. They may be far from optimized for the designated objectives. We use Algorithm 3 to optimize these plans. Take the following allocation plan P for example, where the total resources of the virtual machine set $\{vm_1, vm_2, vm_4\}$ allocated to the test task T_1 are more than required. To optimize the virtual machine resources and avoid fragmentation, the algorithm removes vm_2 which provides the minimum resources in the virtual machine set to optimize plan P without violating the basic resource constraint (steps 5-8 of Algorithm 3).

$$P = \begin{bmatrix}\{vm_1, vm_2, vm_4\}_{T_1}\\ \{vm_3, vm_4, vm_5\}_{T_2}\\ \{vm_5, vm_6, vm_7\}_{T_3}\end{bmatrix}$$
$$\begin{cases} R_{T_1}(vm_1)+R_{T_1}(vm_2)+R_{T_1}(vm_4) > R(T_1)+R_{T_1}(vm_2)\\ R_{T_2}(vm_3) + R_{T_2}(vm_4) + R_{T_2}(vm_5) > R(T_2)\\ R_{T_3}(vm_5) + R_{T_3}(vm_6) + R_{T_3}(vm_7) > R(T_3)\end{cases}$$

Algorithm 3 The Optimization of a Resource Allocation Plan

```
Input: A resource allocation plan $P$
Output: An optimized resource allocation plan $P$
1: for each load testing task $T_k$ in $P$ do
2:   let $VM_k$ be the virtual machine set allocated to $T_k$;
3:   calculate the total available resources in $VM_k$, $R_k = \sum_{vm \in VM_k} R_{T_k}(vm)$, where $R_{T_k}(vm)$ represents the available resources allocated to $T_k$ in $vm$;
4:   select a virtual machine $vm_i$ with the minimum available resources in $VM_k$;
5:   while $R_k - R(T_k) \geq R_{T_k}(vm_i)$ do
6:     remove $vm_i$ from $VM_k$, and update $R_k$ and the available resources of $vm_i$;
7:     select another virtual machine $vm_i$ with the minimum available resources from set $VM_k$;
8:   end while
9: end for
10: return $P$;
```

$$\rightarrow \left\{ \begin{array}{l} \{vm_1, vm_4\}_{T_1} \\ \{vm_3, vm_4, vm_5\}_{T_2} \\ \{vm_5, vm_6, vm_7\}_{T_3} \end{array} \right\}$$

$$\left\{ \begin{array}{l} R_{T_1}(vm_1) + R_{T_1}(vm_4) > R(T_1) \\ R_{T_2}(vm_3) + R_{T_2}(vm_4) + R_{T_2}(vm_5) > R(T_2) \\ R_{T_3}(vm_5) + R_{T_3}(vm_6) + R_{T_3}(vm_7) > R(T_3) \end{array} \right.$$

D. MULTI-OBJECTIVE OPTIMIZATION

In steps 9-10 of Algorithm 1, we mix the parent and child populations and preserve the optimized resource allocation plans to create a new generation of the population so that the populations can evolve toward the objectives. The process of the mixing and creation is shown in Fig. 8.

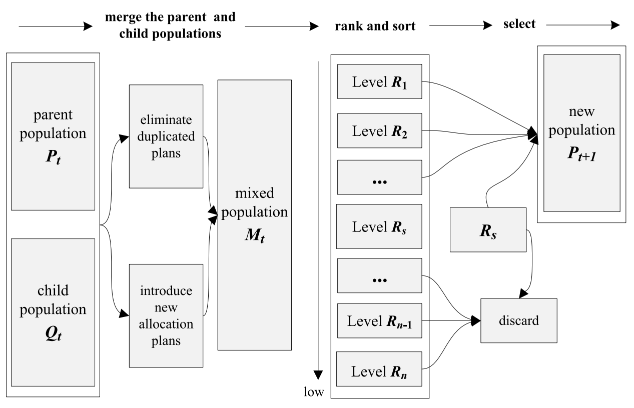


FIGURE 8. The generation of a new population.

In Fig. 8, we merge the parent population P_t and the child population Q_t generated in previous steps into a mixed population M_t. Since the probability of crossover and mutation is usually less than 1, some allocation plans in the parent population P_t will not participate in crossover, mutation, and other operations. They will exist in both the parent population P_t and the child population Q_t. There will be some duplicated plans in the mixed population M_t. This may affect the efficiency of the evolutionary search. We first eliminate the duplicated allocation plans in M_t and introduce new allocation plans to ensure the size of the population M_t.

This work follows the skeleton of the NSGA II (Non-dominated Sorting Genetic Algorithm II) [31] to achieve multi-objective optimization. First, we rank the resource allocation plans in the mixed population M_t into different levels according to the Pareto dominance [32] between them (A Pareto dominates B if A is superior to B in at least one objective and not worse than B in other objectives). The more optimized dominator plans are ranked at higher levels. Then, we adopt the crowding distance [33] (the crowding distance of a resource allocation plan is the sum of the differences between its two adjacent allocation plans on each objective function) to sort the resource allocation plans at the same level. The larger crowding distance, the better for ensuring the diversity of the population. Finally, we select the first N allocation plans from the mixed population M_t to form the new population. The ranking, sorting, and selection are detailed as follows.

1) RANK

First, calculate the dominance relationship between every two allocation plans (P_i, P_j) in the mixed population M_t. We then find the number of allocation plans that dominate a plan P, i.e., n_P, add the allocation plans with $n_P = 0$ to the dominance level R_1, and remove them from M_t. Next, we update the number of allocation plans dominating a plan in the remaining plan set M_t, continue to find the plans with $n_P = 0$, and add them to level R_2. The process is repeated until the entire mixed population M_t is ranked into levels $R_1, R_2, \ldots, R_n$.

2) SORT

Let the objective functions of resource redundancy, test execution cost, and network communication cost be f_1, f_2, and f_3, respectively. At the same dominance level, for each objective function f_k ($k = 1, 2, 3$), we arrange the resource allocation plans in ascending order of their objective function values and calculate

$$L_k(P_i) = (f_k(P_{i+1}) - f_k(P_{i-1}))/(f_k^{\max} - f_k^{\min})$$

for each allocation plan P_i, on objective f_k. The value $L(P_i) = \sum_{k=1}^{3} L_k(P_i)$ is taken as the crowding distance of an allocation plan P_i at the dominance level. We sort the resource allocation plans at the same level in ascending order of the crowding distance.

3) SELECT

We select the first N allocation plans from the mixed population M_t to form an optimized new population from high to low dominance levels and large to small crowding distances in the ranked and sorted results.

TABLE 1. Virtual machine and physical machine settings.

virtual machine	number of CPU cores	1, 2, 3, 4, 5, 6, 7, 8	available CPU resources	(0, 8)
	memory(GB)	{2, 4, 6, 8, 10, 12, 14, 16}	available memory resources	(0, 16GB)
	network bandwidth (Mbps)	{100, 200, 400, 600, 800, 1000}	available network resources	(0, 1000Mbps)
physical machine	number of virtual machines	[4, 16]		

TABLE 2. Cloud testing environment configuration.

environment	#physical machines	#virtual machines	#tasks in the sliding window	#routers
$S1$	109	654	20	207
$S2$	207	1242	40	402
$S3$	313	1878	60	671
$S4$	400	2400	80	823
$S5$	501	3006	100	995

VI. EVALUATION

We validated the effectiveness of the proposed approach by conducting experiments to answer the following research questions.

RQ1: How does the shared-mode resource allocation method compare with an exclusive-mode one in terms of the economy of resource utilization and the efficiency of resource allocation?

RQ2: Does the shared-mode resource allocation affect the effective execution of each load testing task?

RQ3: What are the effects of different parameters, such as the maximum evolution generations, the population size, and the crossover and mutation probabilities, on the results of the resource allocations?

A. EXPERIMENTAL SETUP

We run the resource allocation algorithms on a machine with an Intel(R) i5-7300HQ CPU and 8GB memory to collect the experimental data. For the experimental subjects, a real cloud testing environment can be huge and costly to be used for conducting experiments. We evaluated the performance of resource allocation algorithms (RQ1 and RQ3) on simulated cloud testing environments. The simulated environments are randomly generated. First, we randomly generate a set of physical machines and routers. For each physical machine, a number of virtual machines will be randomly created and bound to the physical machine. Then, we select a router r as the base node and randomly associate physical machines and routers to r to build network connections. The routers connected to r are recursively processed. Finally, we do a breadth-first search to collect the topological structure of the entire network. In the simulation, we assume that the virtual machines used for generating workloads and the virtual machines hosting web services are located on different physical machines.

We generated cloud testing environments of different scales for validation, whose configurations are shown in Table 2, where the scales of the cloud environments increase from $S1$ to $S5$. The configurations and available resources of the virtual machines in a cloud testing environment were randomly generated according to the settings in Table 1. In the table, the available memory resource of 2GB means there is 2GB memory available for the test tools to generate workloads. The total memory in the corresponding virtual machine can be just 2GB or over 2GB. The meanings of the available CPU and network resources are similar. For different scales of the cloud testing environments, the numbers of submitted load testing tasks in a sliding window may vary from small to large. We tested different scales of the numbers of tasks in the sliding windows to evaluate the ability of the resource allocation algorithm in handling different amounts of load testing tasks.

For each cloud testing environment in Table 2, the experiment randomly generated load testing tasks for the sliding window for resource allocation. The configuration of the test tasks is shown in Table 3. The test tasks are driven by eight candidate test scripts with resource consumption ranging from small to large ($s_1 \rightarrow s_8$). The relation between the resource consumption and the load scale of each test script is assumed to be linear. The test duration is randomly generated from the interval (0, 1800s]. For each test script, there are four candidate load change strategies: ramp-up, linear increasing, step up and down, or bell curve.

TABLE 3. Load testing task configuration.

load testing task	test duration	(0, 1800s]
	maximum load scale	(0, 100000]
	candidate scripts	$s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$
	test script resource consumption model	linear function

The parameter settings of the shared-mode resource allocation algorithm are listed in Table 4. We use the population size N, the maximum evolution generation K, etc. to control the multi-objective evolution, and we use the resource cost vector ω_2 and the cost estimation function $Z(TS, P)$ in Section IV-B(2) to determine the test execution cost.

TABLE 4. Parameters of the shared-mode resource allocation algorithm.

parameter	description	value
N	population size	200
K	maximum evolution generation	100
p_c	crossover probability	0.7
p_m	mutation probability	0.1
ω_1	resource weight vector	(0.2, 0.1, 0.1)
ω_2	resource cost vector	(0.2, 0.1, 0.05)
$delay$	time delay of router	0.5

B. RESULTS AND DISCUSSION

1) RQ1: COMPARISON WITH THE EXCLUSIVE-MODE RESOURCE ALLOCATION

For RQ1, we evaluate the economy of resource utilization from the aspects of the resource redundancy, the test execution cost, the network communication cost, and the number of allocated virtual and physical machines. The efficiency of resource allocation is evaluated by the algorithm execution time. On a group of cloud testing environments shown in Table 2, we did 10 rounds of testing with different load testing task sequences executed on each environment to validate the effectiveness of the resource allocation. Resource allocation plans are generated for the same cloud testing environments and test task sequences with both the shared-mode resource allocation method and an exclusive-mode resource allocation method to make comparisons between them.

For the five cloud testing environments, the average results of different allocation methods under the 10 round of task sequences are shown in Table 5. Columns `RR`, `TC`, and `CC` list the resource redundancy, test execution cost, and the network communication cost, respectively (values of the objective functions in Section IV-B). Columns `VM` and `PM` show the numbers of the virtual machines and physical machines occupied by the load testing tasks. Column `t` lists the algorithm execution time.

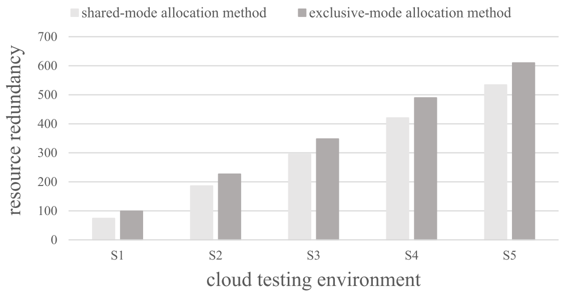


FIGURE 9. Resource redundancy under different methods.

Fig. 9-11 show the average resource redundancy, test execution cost, and network communication cost of the shared-mode and exclusive-mode resource allocation methods on cloud testing environments $S1$ to $S5$ in bar charts. The horizontal axes of these figures list the cloud testing environments, and the vertical axes show the metric values. Compared with the exclusive-mode one, the shared-mode

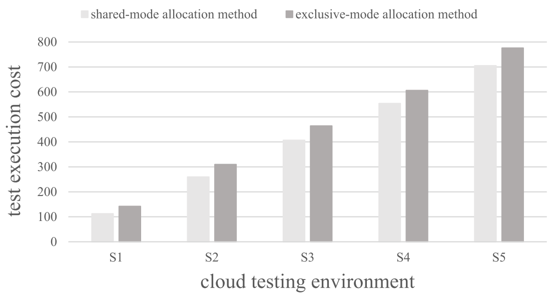


FIGURE 10. Test execution cost under different methods.

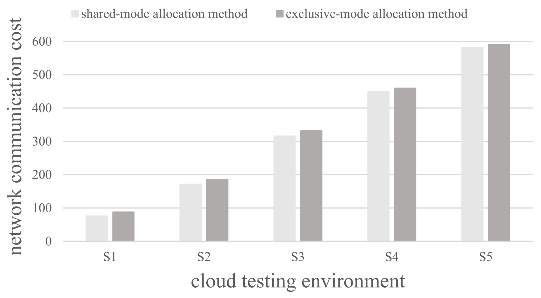


FIGURE 11. Network communication cost under different methods.

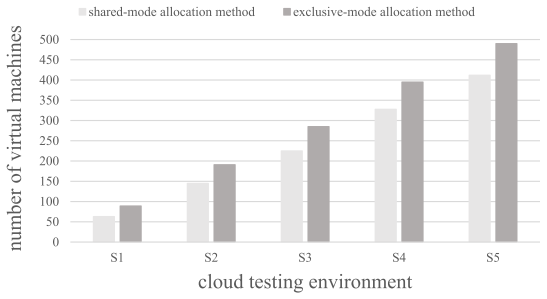


FIGURE 12. The number of virtual machines occupied under different methods.

resource allocation method has less resource redundancy (reduced by 12.43-25.11% on $S_1 - S_5$), lower test execution cost (reduced by 8.63-20.81% on $S1-S5$), and lower network transmission cost (reduced by 1.36-7.58% on $S1 - S5$).

Fig. 12 and 13 show the average numbers of virtual machines and physical machines occupied by the shared-mode and exclusive-mode resource allocation methods on different cloud testing environments. Compared with the exclusive-mode one, the shared-mode resource allocation method occupies fewer virtual machines and physical machines. On the five cloud testing environments $S1$ to $S5$, as the average results of 10 rounds of testing, the numbers of occupied virtual machines were reduced by 26, 46, 60, 67, and 78 with reduction rates 15.92%-29.21%, and the numbers of occupied physical machines were reduced by 16, 20, 22, 27, and 39 with reduction rates 10.23%-25%.

Fig. 14 shows the algorithm execution time of the shared-mode and exclusive-mode resource allocation methods on the cloud testing environments with scales from small to

TABLE 5. Results of the shared-mode and exclusive-mode allocation methods.

allocation method	cloud environment	RR	TC	CC	VM	PM	t (ms)
shared-mode resource allocation	$S1$	74.1	112.56	75.8	63	48	4112
	$S2$	185.93	259.79	171.2	145	110	9184
	$S3$	295.15	406.55	315.8	225	167	16688
	$S4$	420.33	553.86	448.85	328	237	21873
	$S5$	534.22	705.11	582.45	412	288	28340
exclusive-mode resource allocation	$S1$	98.95	142.13	87.7	89	64	3965
	$S2$	226.72	309.79	185.25	191	130	8724
	$S3$	348	464.05	331.7	285	189	16070
	$S4$	489.5	606.15	459.5	395	264	20962
	$S5$	610.07	776.02	590.5	490	327	26939

TABLE 6. Load testing task configurations for RQ2.

	load testing task	maximum load	test duration	web app	test script
load testing task configuration	T_1	8000	120s	Order	view order details, edit order
	T_2	6000	120s	BookManager	search book, view book intro.
	T_3	4000	120s	MailReader	read and add email
task combinations	group	identifier	combined tasks using shared resources		
	single task combination	C_1	$\{T_1\}$		
		C_2	$\{T_2\}$		
		C_3	$\{T_3\}$		
	two task combination	C_4	$\{T_1, T_2\}$		
		C_5	$\{T_1, T_3\}$		
		C_6	$\{T_2, T_3\}$		
	three task combination	C_7	$\{T_1, T_2, T_3\}$		

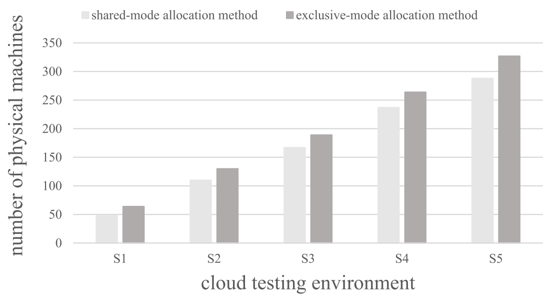


FIGURE 13. The number of physical machines occupied under different methods.

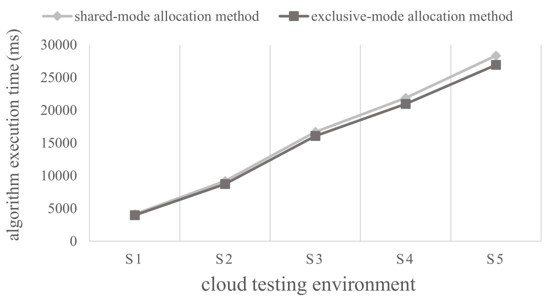


FIGURE 14. Allocation algorithm execution time under different methods.

large. The execution time consumed by these two methods is close (the max difference is 2 seconds) and is all less than 30 seconds. These results show that the shared-mode resource allocation method has similar performance to the exclusive-mode one in terms of resource allocation efficiency. It can generate optimized resource allocation plans for a task sequence in a very short time (compared with the test duration). The resource planning efficiency is high enough to be used in sliding-window-based test task execution.

In summary, the shared-mode resource allocation method performs better than the exclusive-mode resource allocation method in terms of the economy of resource utilization, and the allocation efficiency of the two methods is close.

2) RQ2: EFFECTS ON THE EFFECTIVE EXECUTION OF EACH LOAD TESTING TASK

For RQ2, we run real load testing tasks in the shared-mode on the cloud testing system developed by our laboratory to investigate whether sharing virtual machine resources does not affect the effective execution of load testing tasks. The throughput (the amounts of workloads finished per unit of time) and execution time (the time from execution start to execution end) of load testing tasks are used to evaluate the impacts of resource sharing.

We used a representative virtual machine *VM* with a hardware resource configuration of 4 CPU cores, 8GB memory, and 1000Mbps network as the experimental environment (the hardware resources are approximately just enough to launch the test). As shown in Table 6, three representative web applications `Order`, `BookManager`, and `MailReader` were used as the load testing subjects. `Order` is an order management system used to manage users' order records. We tested viewing and editing order operations in load testing task T_1. `BookManager` is a book management system.

We tested searching for a book and view book introduction operations in load testing task T_2. `MailReader` is an email system. We tested reading and adding email operations in load testing task T_3. The load testing tasks T_1, T_2, and T_3 linearly increase the workloads until reaching the maximum load. They are executed in seven combinations (C_1 to C_7) to investigate the impacts of resource sharing. In each combination, the load testing tasks are simultaneously executed on the virtual machine VM. The experimental results are shown in Table 7, where tps represents the throughput and $t_{execute}$ represents the actual execution time of a test task.

TABLE 7. Execution results for different task combinations.

	single task combination	two task combination		three task combination
T_1	C_1	C_4	C_5	C_7
tps	433.37	429.78	433.37	422.80
$t_{execute}$	120	121	120	123
T_2	C_2	C_4	C_6	C_7
tps	330.51	325.00	329.73	322.31
$t_{execute}$	118	120	119	121
T_3	C_3	C_5	C_6	C_7
tps	216.70	214.91	214.91	213.15
$t_{execute}$	120	121	121	122

The results in Table 7 show that for load testing task T_1, when being separately executed (exclusive-mode resource allocation), according to the single task combination C_1, its tps is 433.37, and the $t_{execute}$ is 120s. When sharing resources in the virtual machine VM with other tasks T_2 and T_3, according to multiple task combinations C_4, C_5, and C_7, the tps is 429.78, 433.37, and 422.80, and the $t_{execute}$ is 121s, 120s, and 123s. The throughput and text execution time of task T_1 is close under different execution combinations. The results on load testing tasks T_2 and T_3 are similar. We can see that, when there are sufficient resources, whether a load testing task occupies an entire virtual machine or shares the virtual machine resources with other tasks does not affect the effective execution of the task. The shared-mode resource allocation can be used in effective load testing.

3) RQ3: EFFECTS OF DIFFERENT PARAMETER SETTINGS

We used S_1 in Table 2 as a representative cloud testing environment and run the shared-mode resource allocation algorithm on S_1 under different parameter settings for the load testing task sequences in the sliding windows to evaluate the effects of different parameter value choices.

Fig. 15 shows the results of the multi-objective algorithm under different maximum evolution generation settings. The x-axis of the figure represents the maximum evolution generations. The y-axis represents the values of the objective functions and the execution time of the algorithm. For the same load testing task sequences with resources to be allocated, when the maximum evolution generation increases, the resource redundancy, the test execution cost, and the network communication cost of the generated allocation plan go down and tend to be more optimized. The algorithm execution

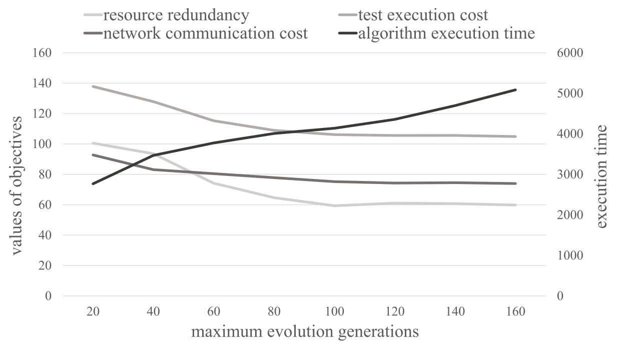


FIGURE 15. The impacts of the maximum evolution generations.

time gradually increases. This suggests that increasing the maximum evolution generation is beneficial for achieving more optimization. However, when the maximum evolution generation is too high, the optimization reward grows slow, but the algorithm execution time increases fast. Too high maximum evolution generation might not be economical. Therefore, we set the maximum evolution generation to 100 in answering RQ1.

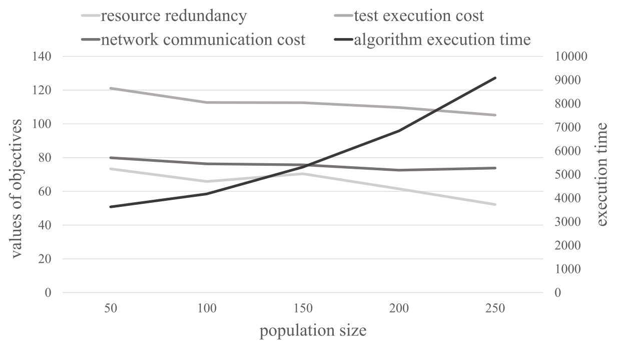


FIGURE 16. The impacts of the population sizes.

Fig. 16 shows the results of the algorithm under different levels of population sizes. For the same load testing task sequences to be handled, when the population size increases, the resource redundancy, the test execution cost, and the network communication cost of the generated allocation plan exhibit a downward trend. The greater the population size, the more optimization toward the objectives. However, as the population size increases, the execution time of the algorithm grows up in nearly a power function ($x^{\alpha}, \alpha > 1$). The population size has a significant impact on the efficiency of the resource allocation algorithm. In the experiments, we used a population size of 200, which provides a good trade-off between algorithm efficiency and resource allocation optimization, to answer RQ1.

Fig. 17 shows the results of the algorithm under different crossover probabilities. For the same load testing task sequences to be processed, when the crossover probability increases, the resource redundancy and the test execution cost of the generated allocation plan grow down, and network

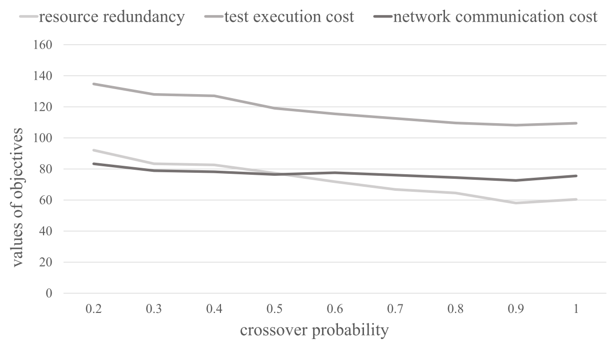


FIGURE 17. **The impacts of the crossover probabilities.**

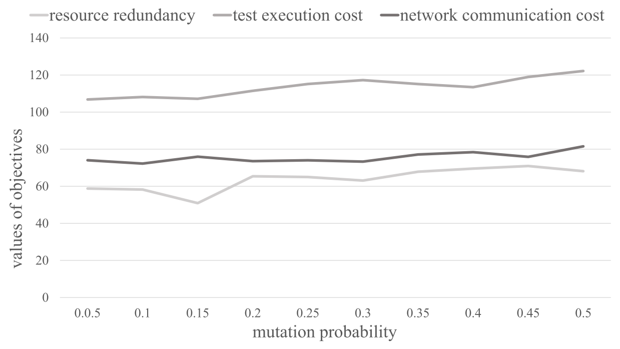


FIGURE 18. **The impacts of the mutation probabilities.**

communication cost seems to be almost stable. In general, increasing the crossover probability looks beneficial for the optimization objectives. Therefore, we set a high crossover probability of 0.7 for RQ1.

Fig. 18 shows the results of the algorithm under different mutation probabilities. From the figure, we can see that the resource redundancy, the test execution cost, and the network communication cost of the generated allocation plan are better when the mutation probability is in [0.05, 0.15]. This suggests that the mutation probability should better not be too large. Too large mutation probability may make the algorithm tend to do random searches, which cannot guarantee the evolution of the allocation plans toward more optimized directions. According to the above analysis, we set the mutation probability to 0.1 for RQ1.

C. THREATS TO VALIDITY

There are three major validity threats in the experiments. (1) The first is in the parameter settings of the genetic algorithm. Different parameter settings may lead to different results. Nevertheless, we determined the parameter values by experiments and listed the main parameters in Section VI-A. We believe these parameter settings are reasonable. (2) The second is in the cloud testing environments used in the experiments, which may limit the generalization of the experimental results. We evaluated the approach on a group of cloud testing environments with sizes from small to large. This demonstrates the effectiveness of the approach on different scales of the clouds. Although the cloud environments are simulated, we simulated the features key to test resource allocation, which might be enough for drawing some experimental conclusions. (3) The third is in the number of tasks in the sliding window. The experimental results may vary under different numbers of tasks. Even so, we tested different scales of tasks in the windows. This can relieve the impacts of the task sequence sizes on the experimental conclusions.

VII. CONCLUSION

The paper presents a shared-mode resource allocation method for cloud-based load testing to more economically use virtual machine resources in the cloud. We firstly introduce a multi-objective (objectives of minimizing resource redundancy, minimizing test execution cost, and minimizing network communication cost) and constrained (the minimum resource requirements must be meet) shared-mode resource allocation model that can save more test resources. Based on the model, we propose a shared-mode resource allocation algorithm to allocate test resources for a sequence of cloud-based load testing tasks coming in a time window. The experimental results show that compared with an exclusive-mode resource allocation method, the proposed method effectively reduced the resource redundancy, test execution cost, and network communication cost. It saved more than 15% virtual machines and more than 10% physical machines for cloud-based load testing. This indicates that the method might be valuable for practical load testing.

REFERENCES

[1] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Trans. Softw. Eng.*, vol. 41, no. 11, pp. 1091–1118, Jun. 2015.

[2] T.-H. Chen, M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Analytics-driven load testing: An industrial experience report on load testing of large-scale systems," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng., Softw. Eng. Pract. Track (ICSE-SEIP)*, Buenos Aires, Argentina, May 2017, pp. 20–28.

[3] H. Malik, H. Hemmati, and A. E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, May 2013, pp. 1012–1021.

[4] L. Riungu-Kalliosaari, O. Taipale, K. Smolander, and I. Richardson, "Adoption and use of cloud-based testing in practice," *Softw. Qual. J.*, vol. 24, no. 2, pp. 337–364, Jun. 2016.

[5] L. M. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Indianapolis, IN, USA, Nov. 2010, pp. 557–564.

[6] J. Noor, M. G. Hossain, M. A. Alam, A. Uddin, S. Chellappan, and A. B. M. A. Al Islam, "SvLoad: An automated test-driven architecture for load testing in cloud systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–7.

[7] A. Bertolino, G. D. Angelis, M. Gallego, B. García, F. Gortázar, F. Lonetti, and E. Marchetti, "A systematic review on cloud testing," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 93:1–93:42, Sep. 2019.

[8] T. Parveen and S. Tilley, "When to migrate software testing to the cloud?" in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Paris, France, Apr. 2010, pp. 424–427.

[9] Q. Gao, W. Wang, G. Wu, X. Li, J. Wei, and H. Zhong, "Migrating load testing to the cloud: A case study," in *Proc. IEEE 7th Int. Symp. Service-Oriented Syst. Eng.*, San Francisco, CA, USA, Mar. 2013, pp. 429–434.

[10] W. Kang, J. Xiao, and X. Kong, "Research on cloud test resource allocation based on improved fuzzy clustering PSO algorithm," *J. Phys., Conf. Ser.*, vol. 1176, Dec. 2018, Art. no. 022005.

[11] P. Lampe and J. Rudy, "Models and scheduling algorithms for a software testing system over cloud," in *Proc. 13th Int. Conf. Dependability Complex Syst.*, Brunów, Poland, May 2018, pp. 326–337.

[12] H. Lu, R. Niu, J. Liu, and Z. Zhu, "A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2790–2802, May 2013.

[13] *WeTest: Tencent Performance Testing Service*. Accessed: Aug. 4, 2020. [Online]. Available: https://wetest.qq.com/product/gaps

[14] *PTS: Alibaba Performance Testing Service*. Accessed: Aug. 4, 2020. [Online]. Available: https://www.aliyun.com/product/pts

[15] H. Goudarzi and M. Pedram, "Maximizing profit in cloud computing system via resource allocation," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. Workshops*, Minneapolis, MN, USA, Jun. 2011, pp. 20–24.

[16] K. Agrawal and S. Baruah, "A measurement-based model for parallel real-time tasks," in *Proc. Euromicro Conf. Real-Time Syst.,* Barcelona, Spain, Jul. 2018, pp. 5:1–5:19.

[17] L. Cano, G. Carello, and D. Ardagna, "A framework for joint resource allocation of MapReduce and Web service applications in a shared cloud cluster," *J. Parallel Distrib. Comput.*, vol. 120, pp. 127–147, Oct. 2018.

[18] Z. Zhu and Z. Du, "Improved GA-based task scheduling algorithm in cloud computing," *Comput. Eng. Appl.*, vol. 49, no. 5, pp. 77–80, 2013.

[19] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, pp. 63:1–63:33, Jul. 2015.

[20] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019.

[21] B. Keshanchi, A. Souri, and N. J. Navimipour, "An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing," *J. Syst. Softw.*, vol. 124, pp. 1–21, Feb. 2017.

[22] Y. Sun, J. White, S. Eade, and D. C. Schmidt, "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization," *J. Syst. Softw.*, vol. 116, pp. 146–161, Jun. 2016.

[23] T. Aladwani, "Impact of tasks classification and virtual machines categorization on tasks scheduling algorithms in cloud computing," in *Proc. 2nd Int. Conf. Internet Things, Data Cloud Comput. (ICC)*, Cambridge, U.K., Mar. 2017, p. 158:1–158:7.

[24] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[25] JMeter. *Apache JMeter*. Accessed: Aug. 4, 2020. [Online]. Available: https://jmeter.apache.org/

[26] Selenium. *Selenium Browser Automation*. Accessed: Aug. 4, 2020. [Online]. Available: https://www.selenium.dev/

[27] FunkLoad. *FunkLoad: Functional and Load Web Tester*. Accessed: Aug. 4, 2020. [Online]. Available: https://pypi.org/project/funkload/

[28] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evol. Comput.*, vol. 3, no. 1, pp. 1–16, Mar. 1995.

[29] *ECS: Elastic Compute Service*. Accessed: Aug. 4, 2020. [Online]. Available: https://www.alibabacloud.com/product/ecs

[30] A. Hiley and B. A. Julstrom, "The quadratic multiple knapsack problem and three heuristic approaches to it," in *Proc. 8th Annu. Conf. Genetic Evol. Comput. (GECCO)*, Seattle, WA, USA, Jul. 2006, pp. 547–552.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[32] Y. Lei, M. Gong, J. Zhang, W. Li, and L. Jiao, "Resource allocation model and double-sphere crowding distance for evolutionary multi-objective optimization," *Eur. J. Oper. Res.*, vol. 234, no. 1, pp. 197–208, Apr. 2014.

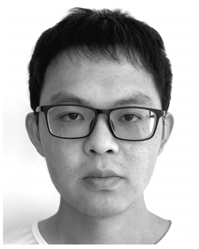
[33] A. Metiaf, Q. Wu, and Y. Aljeroudi, "Searching with direction awareness: Multi-objective genetic algorithm based on angle quantization and crowding distance MOGA-AQCD," *IEEE Access*, vol. 7, pp. 10196–10207, Jan. 2019.

WENMING JIN received the B.Sc. degree in computer science from the Anhui University of Finance and Economics, Bengbu, China, in 2017. He is currently pursuing the M.S. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

His research interest includes cloud-based software testing.

JU QIAN (Member, IEEE) received the B.Sc. and Ph.D. degrees in computer science from Southeast University, Nanjing, China, in 2003 and 2009, respectively.

He is currently an Associate Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing. His research interests include program analysis, software testing, debugging, and evolution.



SHUOYAN YAN received the B.Sc. degree in computer science from the China University of Mining and Technology, Xuzhou, China, in 2016. He is currently pursuing the M.S. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

His research interests include cloud-based software testing and test automation.