Adaptive Request Scheduling for Device Cloud

Han Dong[1], Enze Xu[2], *Xiang Jing[1], Huaqian Cai[2], and Gang Huang[2]

[1]School of Software and Microeletronics, Peking University, Beijing, China
[2]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
Email: [1,2]{danieldong, xez, jingxiang, caihq, hg}@pku.edu.cn

***Abstract*—Nowadays, more and more cloud testing platforms provide enterprise developers with solutions for cloud device debugging and automatic testing. It is a great challenge for these cloud platforms to schedule the arriving requests to run on the specific smart device resources in real-time and efficiently. The traditional scheduling algorithm is difficult to adapt to the application interface call request with a vast difference in volume and behaviour ability. To solve this problem, we integrate these smart devices into a device cloud and propose a measurement method of the service capability of a single device in the group. Then we build an adaptive scheduling algorithm model according to the characteristics of the serviceability of a single device to improve the scheduling efficiency of the group. Practice shows that the adaptive scheduling algorithm can effectively control the network traffic. Finally, through the analysis and optimization, we get the method of obtaining the optimal parameter combination in the algorithm.**

***Keywords*-*adaptive scheduling; smart devices; traffic control; device cloud; competitive counting;**

I. Introduction

The use of smartphone has substantially increased in the past decade. It is estimated that by the end of 2020, the number of smartphone users in the world will be 6.1 billion [1]. In the era of smart device Internet, more and more cloud testing platforms provide enterprise developers with solutions for smart cloud device debugging and automatic testing [2].

It is a great challenge for these cloud platforms to schedule the arriving requests to run on the specific smart device resources in real-time and efficiently. The traditional scheduling algorithm is difficult to adapt to the application interface call request with a vast difference in volume and behaviour ability.

In previous studies, the adaptive scheduling algorithm is mainly used in the work of network signal and transmission. For example, the dynamical adjusting of the inter-frame space (IFS) is based on differences between required transmission delay and estimated experienced transmission delay [3]; The adaptive sampling rate scheduling (ASRS) methodology maintains the transmitted measurement signals' fidelity [4], provides a maximum level of adaptability accommodating packet losses and changes in network topology while expanding periodic nature of the sensor node transmissions [5]. These service scheduling algorithms rarely pay attention to the enormous service performance differences before different service types, mainly taking the quantitative indicators of actual traffic as the adaptive data reference.

If there is a smart device scheduling algorithm that can dynamically balance the load of each device in the device group, such scheduling algorithm can greatly improve the overall scheduling efficiency of a device group. The application of this algorithm in cloud testing platforms also has significant economic value.

A commonly used Android device testing method is implemented by running the virtual machine through Docker [6] installed on the Linux operating system. But in our experiment, we find that the stability of Android system in Docker is much worse than that installed directly on smart devices. So in this paper, we directly use the physical Android system on smart devices for testing.

Our previous experiment concludes that with the increase of the arrival rate of call requests, the percentage of successfully responding to and running the same application request on a single device match a model of "idle-saturation-overload". There is a fixed optimal load rate [7] for the specified device and application request. Under this loading rate, the throughput per unit time of the devices is the largest, and the service performance is the best. However, in a real application scenario, it is difficult for us to calculate the "optimal load rate" of an unknown application request in advance, because the calculation of this value itself requires a large number of tests to calibrate, which is difficult for a busy platform server to accept.

We design an adaptive scheduling algorithm to solve this problem. We integrate these smart devices into a device cloud and propose a measurement method of the service capability of a single device in the group. Then we build an adaptive scheduling algorithm model according to the characteristics of the serviceability of a single device to improve the scheduling efficiency of the group. Our experiment shows the adaptive scheduling algorithm can effectively control the network traffic and help the task centre of the group to decide the future task scheduling according to the success of each application request. Finally, through the analysis and optimization, we get the method of obtaining

*corresponding author: jingxiang@pku.edu.cn the optimal parameter combination in the algorithm.

II. Background and Related Works

In this section, we give some background knowledge about the framework of behavioural reflection of Internetware and the scheduling algorithm of the distributed system.

A. *Framework of Behavioural Reflection of Internetware*

Internetware is an abstract form of the software system in the open, dynamic, and changeable environment of the Internet [8]. It has the characteristics of autonomy, cooperation, responsiveness, evolution, and polymorphism [8], [9]. Android applications are typical Networking software. The "Framework of behavioural reflection of Internetware" proposed by the Software Engineering Institute, Peking University, can open the internal data of Android Application in the form of a "behavioural reflection interface" for external access [10]. Compared with the traditional data open method, the Network Architecture Software Behaviour Reflection Technology Framework has the following advantages:

- **Advantage 1**. It applies to almost all Android apps and does not need the source code of the app or the assistance of the app developers.
- **Advantage 2**. The data obtained is completely consistent with the data in the application program.
- **Advantage 3**. Real-time data within the app can be obtained.
- **Advantage 4**. It does not affect the normal operation of the app program, and the app also has no awareness of data opening.

B. *Scheduling Algorithm of The Distributed System*

Distributed system [11] is a system which is composed of components deployed in multiple networking devices and realizes communication and cooperation between components through message passing. The mechanism and strategy of managing access of consumers to resources in a distributed system are collectively referred to as the scheduling of a distributed system [12]. The scheduling algorithms of the distributed system can be classified according to different standards [12]: static scheduling and dynamic scheduling, centralized scheduling and distributed scheduling, optimal scheduling, and sub-optimal scheduling, primary distribution scheduling and dynamic redistribution scheduling, load-balanced scheduling and non-load-balanced scheduling, adaptive scheduling, and non-adaptive scheduling.

Due to the apparent difference between the device cloud based on the framework of behavioural reflection of Internetware and the general distributed system, the existing distributed system scheduling algorithm is not suitable for the device cloud.

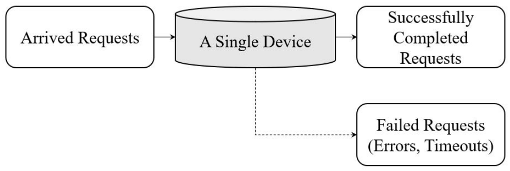

Arrived Requests
A Single Device
Successfully Completed Requests
Failed Requests (Errors, Timeouts)

Figure 1. Request Traffic of Single Device in the Device Cloud

III. Measurement of Service Capability

In this section, we measure the service capability of different action reflective interfaces on a single device under various request traffic.

A. *Definition of Service Capability*

The service capability of the classical data service group is usually expressed by the response time, throughput, and other indicators. The typical application scenario of the device cloud is data batch processing, which is insensitive to response time, so only the throughput needs to be considered for the definition of service capability.

Figure 1 is a schematic diagram of the request traffic of a single device in the device cloud. The number of requests completed per unit time is the throughput of a single device. Since the throughput of a single device is related to the rate of request arrival, the service capability of a single device can be defined as a function $F_i(v)$, where i is the behavioural reflection interface for request invocation, v is the rate of request arrival (in times/seconds), and $F_i(v)$ is the rate of request completion (in times/seconds). The results of function $F_i(v)$ with different values of v can be measured by experiments (see III.B).

B. *Measuring Method*

There are two steps to measure service capability:
Step 1: Control variables

- **Hardware and software configuration of the device**. Measurements are made on the same manufacturer, the same model, and the same Android version of the device.
- **Network environment**. The device is connected to a stable WiFi access point and measured only in non-peak time.

Step 2: Measure the service capability function $F_i(v)$ of a single device (see III.A for definition)

A request to call interface i is sent to the device at rate $v = v_0$, and the rate w of request completion is measured. Repeat several times and get the mean $\overline{\omega}$ of w, then $F_i(v_0) = \omega$ Similarly, the results of $F_i(v)$ with other values of v can be measured. Because the operation of measuring a single $F_i(v)$ is time-consuming and the value range of v is large, if v is not selected properly, the total measurement time will be unbearable. To determine the overall trend, maximum value and variation law near the maximum value of $F_i(v)$ in a

Table I
MEASURED BEHAVIOURAL REFLECTION INTERFACES

Application	Interface	Parameters
App 1	Search audio	Keyword
App 2	Get new product list	/
App 3	Search video	Keyword
App 4	Search news	Keyword
App 5	Search stock information	Stock name or code
App 6	Search for preferential information	Keyword
App 7	Get scenic spots recommendation	City code

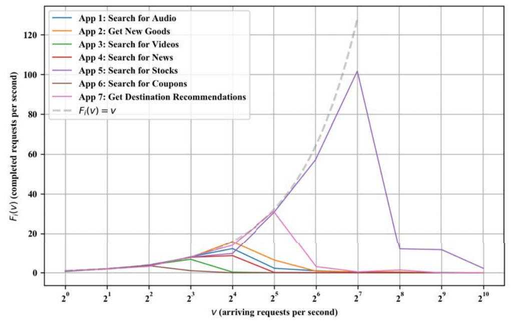


Figure 2. Measurement Results of Service Capability of Some Interfaces of a Single Device (Overall Trend)

reasonable time, the following methods should be used to select v:

- Firstly, v grows exponentially (e.g. $v = 1, 2, 2^2, 2^3, \ldots, 2^k$) to determine the overall trend of $F_i\left(v\right)$ and the interval where the maximum value is located.
- Then, v linearly traverses the interval where the maximum value is located to determine the maximum value of $F_i\left(v\right)$ and the variation law of $F_i\left(v\right)$ near the maximum value.

C. Measurement Results of Partial Interfaces

In this subsection, we use the method described in III.B to measure the service capability of some behavioural reflection interfaces on a single device. The experimental equipment is the "Changhong S07" mobile phone, and the Android version is 6.0. We install BusyBox[13] (command-line tools provided by BusyBox are required for some operations) and Xposed Framework[14] (Xposed Framework is required for target application to load behaviour reflection interface) on these mobile phones. The selected interfaces come from seven typical applications (see Table I).

Figure 2 is a measurement of v with exponential growth, showing the overall trend of $F_i\left(v\right)$ (as defined in III.A):

- With the increase of v, $F_i\left(v\right)$ increases first and then decreases, and finally approaches 0. The change of $F_i\left(v\right)$ accords with the typical process of "idle-saturation-overload".
- If $v_i^* = argmaxF_i\left(v\right)$, then when $v < v_i^*$ (i.e. growth stage), $F_i\left(v\right)$ can be approximated by $F_i\left(v\right) = v$ (gray dotted line), indicating that the interface can provide services more steadily before the requested traffic is saturated.
- The maximum value of $F_i\left(v\right)$ of different interfaces is different and may vary greatly, which reflects the difference of service capability. For example, $\max F_i\left(v\right) < 5$ for the "Search Preferential Information" interface of "Meituan" application, while $\max F_i\left(v\right) > 100$ for the "Search Stock Information" interface of "Yimeng Trader (Classic Edition)". The difference between the two is more than 20 times.

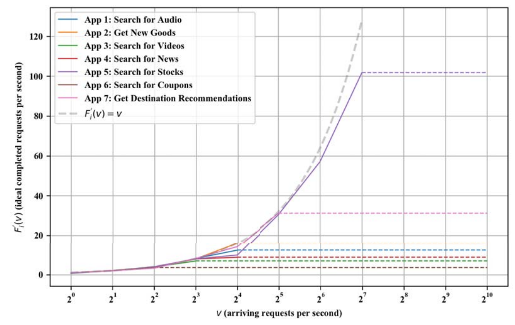


Figure 3. Measurement Results of Service Capability of Some Interfaces of a Single Device (Near the Maximum)

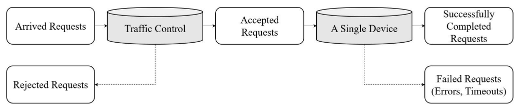


Figure 4. Request Traffic of a Single Device in the Device Cloud

Figure 3 shows the measurement results when v linearly traverses the interval where the maximum $F_i\left(v\right)$ is located. The meanings of abscissa, ordinate, and grey dashed lines are the same as those in Figure 2.

D. Ideal Value under Traffic Control Conditions

From the measurement results of III.C, it can be inferred that if some traffic control mechanism is used to reject some requests when v is too large (see Figure 3), the rate of accepting requests by the device v' satisfies $v' \leq v_i^*$ (v_i^* is the v that maximizes the value of $F_i\left(v\right)$, see III.C), the service capability degradation caused by overload can be avoided.

Define $F_i'\left(v\right)$ as the ideal value of $F_i\left(v\right)$ under traffic control conditions (the image of $F_i'\left(v\right)$ is shown in Figure 5):

$$F_i'\left(v\right) = \begin{cases} F_i\left(v\right), v < v_i^* \\ F_i\left(v_i^*\right), v \geq v_i^* \end{cases} \tag{1}$$

When $v < v_i^*$, $F_i\left(v\right)$ can be approximately expressed as $F_i\left(v\right) = v$ (see III.C), so $F_i'\left(v\right)$ can be approximately

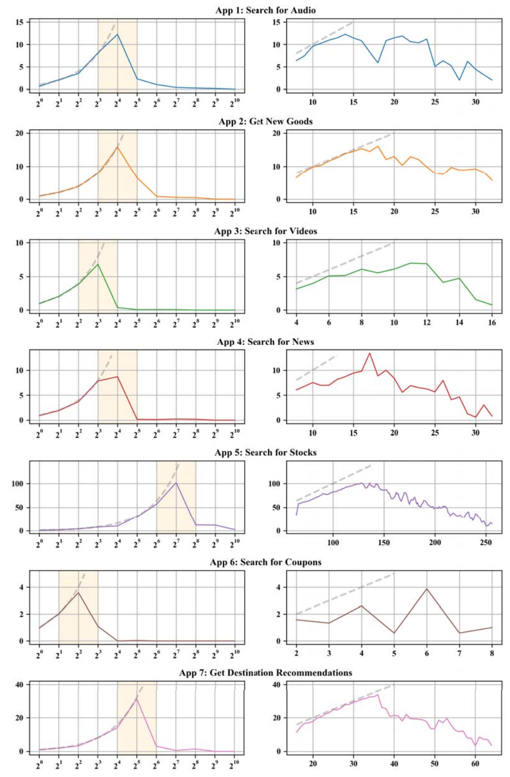


Figure 5. Ideal Value of Partial Interface Service Capability of a Single Device under Traffic Control (Overall Trend)

expressed as $F_i''(v)$ (in IV.B we use this approximation):

$$F_i''(v) = \begin{cases} v, v < v_i^* \\ F_i(v_i^*), v \geq v_i^* \end{cases} \tag{2}$$

IV. Model of Adaptive Scheduling

In this section, we establish an adaptive scheduling model for the device cloud, which is based on the measurement of the service capability of a single device in the last section.

A. Model Assumptions

The device cloud suitable for adaptive scheduling method should satisfy the following assumptions:

- The group consists of n devices with the same hardware and software configuration (manufacturers, models, Android versions are all the same). (No. $1, 2, \ldots, n$).
- The network environment of the group is stable.
- m interfaces (No. $1, 2, \ldots, m$) are deployed. The set of device numbers for deployed interface i is D_i.
- The service capability of the same interface on each device is the same, expressed by the function $F_i(v)$ (defined as III.A).
- Different interfaces deployed on the same device do not influence each other; that is, function $F_i(v)$ has nothing to do with other interfaces deployed on the same device.

The device cloud Q satisfying the above assumptions can be represented by a $2m+2$ tuple:

$$Q = (n, m, D_1, D_2, \ldots, D_m, F_1, F_2(v), \ldots, F_m(v)) \tag{3}$$

B. Scheduling Objective

In III.A we define the service capability $F_i(v)$ of a single device. Similarly, the total service capability of the group Q on interface i can be defined as a function $G_{Q,i}(v)$, where v is the rate at which requests reach the group (in times/seconds) and $G_{Q,i}(v)$ is the rate at which the group completes requests (in times/seconds). The request rate assigned to device j by group gateway is v_j (in times/seconds). Then we get:

$$v = \sum_{j \in D_i} v_j, G_{Q,i}(v) = \sum_{j \in D_i} F_i(v_j) \tag{4}$$

$F_i''(v)$ is an approximate representation of the ideal value of $F_i(v)$ under traffic control conditions (see III.D). It is easy to prove that:

$$G_{Q,i}(v) \leq |D_i| F_i''(\frac{v}{|D_i|}) \tag{5}$$

That is, $|D_i| F_i''(\frac{v}{|D_i|})$ is the upper limit of $G_{Q,i}(v)$ for given $|D_i|$ and $F_i(v)$. Define the scheduling efficiency function $H_{Q,i}(v)$ of interface i on the group Q:

$$H_{Q,i}(v) = \frac{G_{Q,i}(v)}{|D_i| F_i''(\frac{v}{|D_i|})}, 0 < H_{Q,i}(v) \leq 1 \tag{6}$$

Define the integrated scheduling efficiency function $H_Q(v)$ of m interfaces on the group Q:

$$H_Q(v) = \frac{m}{\sum_{i=1}^{m} \frac{1}{H_{Q,i}(v)}}, 0 < H_Q(v) \leq 1 \tag{7}$$

Group parameters (i.e. n, m, D_i, F_i) and the scheduling method can affect $H_Q(v)$. When group parameters remain unchanged, $H_Q(v)$ reflects the performance of scheduling: the larger $H_Q(v)$ is, the better the performance of scheduling is. Therefore, this paper chooses maximizing $H_Q(v)$ as the scheduling objective of the group. We measure and calculate the $H_Q(v)$ with a method similar to III.B.

C. Optimization Method

Formula (7) shows that in order to improve $H_Q(v)$, it is necessary to increase $H_{Q,i}(v)$. It can be seen from Formula (5) and Formula (6) that to improve $H_{Q,i}(v)$, it is necessary to make $G_{Q,i}(v)$ close to the upper limit, which is equivalent to making Formula (5) approximately equal. The same sign condition of Formula (5) is satisfied at the same time:

$$\begin{cases} F_i(v) = F_i''(v) \\ v_j = \dfrac{v}{|D_i|}, j \in D_i \end{cases} \tag{8}$$

Line 1 of Formula (8) requires the traffic of requests to be controlled to ensure that the rate at which each device in D_i receives requests does not exceed the critical value v_i^* (see III.D); Line 2 of Formula (8) requires an average allocation of request traffic so that each device in D_i receives requests at the same rate (i.e., the load is equal). The scheduling method only needs to control and distribute the request traffic properly, and to approximate satisfies Formula (8) to make $G_{Q,i}(v)$ close to the upper limit, then $H_{Q,i}(v)$ close to 1, and finally $H_Q(v)$ close to 1, to achieve the scheduling goal of IV.B. Two schemes can approximate Formula (8) satisfactorily:

- **Sche 1**. Continuously monitor the traffic to each device to ensure that it does not exceed the critical value of v_i^*; according to the principle of load balancing, priority should be given to assigning requests to devices with smaller traffic. This scheme is easy to understand, but there are some difficult problems to solve:
  - a. The critical value v_i^* of different interfaces is different, so it is impossible to measure the critical value of all interfaces by experiment beforehand.
  - b. When the request traffic to the group is heavy, the traffic to each device is confronted with the contradiction of low delay and high accuracy: high accuracy requires higher data sampling frequency, while this will increase the delay.
- **Sche 2**. Adaptively control traffic based on the execution result of the request (success or failure). This scheme does not need to measure the critical value of the interface v_i^*, nor need to monitor the traffic to each device explicitly, so it is feasible.

In this paper, we choose **Sche 2**. The rest of this section and the next section introduce the scheme in detail.

D. Algorithm Framework

The key to adaptive scheduling algorithm is adaptive traffic control (see IV.C). The implementation method is as follows (the meaning of n, m, D_i please see IV.A):

- **Step 1**. Maintain the receiving window matrix $W_{n\times m}$, in which element $W_{j,i}$ is used to record the maximum number of simultaneous calls to interface i on device j. The initial value of $W_{j,i}$ can be set to a smaller integer, and $W_{j,i}$ will be updated continuously during the operation of the algorithm.
- **Step 2**. Maintain the current concurrency matrix $C_{n\times m}$, in which element $C_{j,i}$ record the current concurrency of interface i on device j (the number of requests being processed). When the gateway forwards the request for interface i call to device j, $C_{j,i}$ adds 1. When device j processes a request for interface i call, $C_{j,i}$ decreases by 1 regardless of whether the execution is successful or not.

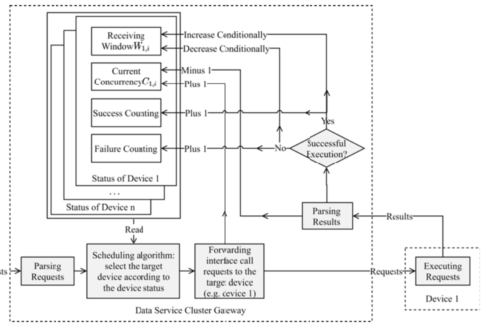


Figure 6. Framework of Adaptive Scheduling Algorithm

- **Step 3**. When the gateway receives the call request of interface i, it tries to find the target device j':

$$\begin{cases} j' = \arg\min_{j\in D_i} \dfrac{C_{j,i}}{W_{j,i}} \\ C_{j',i} < W_{j',i} \end{cases} \tag{9}$$

- **Step 4**. If there is no target device j' meets Formula (9), the current limiting operation will be performed, i.e. to wait for the qualified equipment to appear before it is processed. If there is a timeout in the waiting process, the request will be rejected; if there is a target device j', the request will be forwarded to the device j' for execution. Wait for the execution result.
- **Step 5**. $W_{j,i}$ automatically adjusts according to the results of request execution, according to the following principles: if a large number of requests fail in a short time, reduce $W_{j,i}$; if a large number of requests succeed in a short time, increase $W_{j,i}$; otherwise, $W_{j,i}$ will remain unchanged. The specific rules for adjusting $W_{j,i}$ are shown in IV.E.

The essence of the above methods: indirectly describing the service capability $F_i(v)$ of a single device with $W_{j,i}$ and describing the load of the device with $C_{j,i}$; ensuring that $C_{j,i} \le W_{j,i}$ to achieve traffic control; choosing the smallest equipment with $\dfrac{C_{j,i}}{W_{j,i}}$ to achieve average traffic distribution without precise monitoring the instantaneous request traffic to the device; automatically adjusting $W_{j,i}$ according to the success or failure of the interface call to achieve an adaptive interface with the unknown rate v_i^*.

Figure 6 is an adaptive scheduling algorithm framework based on this method.

E. Algorithm Parameter

In this subsection, a parameterized $W_{j,i}$ adjustment rule is proposed, which can optimize parameters for specific groups to achieve better performance. $W_{j,i}$ adjusts the parameters of the rules, that is, all the parameters of the adaptive scheduling algorithm. In VI.B we discuss the method of determining the parameters. It can be seen from IV.D that two key problems need to be solved in determining $W_{j,i}$ adjustment rules:

- **Prob 1**. How to judge whether a large number of requests succeed or fail in a "short time"
- **Prob 2**. The amount of $W_{j,i}$ should be increased or decreased.

The second problem is not difficult to solve. Because group devices are very sensitive to request overload (see III.C), the adjustment of $W_{j,i}$ should follow the additive-increase multiplicative-decrease (AIMD) [15] algorithm. In the first problem, the judgment of "a large number" can be based on the ratio of the number of successful or failed requests to the traffic of requests arriving at the device. The real difficulty is the definition of "short time". The "competitive counting" rule proposed in this subsection can avoid the direct judgment of "short time" and achieve the desired results. The following are the "competitive counting" rules:

- **Rule 1**. Maintain the current traffic matrix $V_{n\times m}$, in which element $V_{j,i}$ records the rate at which requests to call interface i reach device j. $V_{j,i}$ does not need to be updated frequently (e.g. every minute).
- **Rule 2**. Maintain the successful request counting matrix $S_{n\times m}$. Whenever device j calls interface i successfully, $S_{j,i}$ adds 1. If the updated $S_{j,i}$ satisfies $S_{j,i} \geq \mu V_{j,i}$, then $W_{j,i}$ adds 1 and $S_{j,i}$ and $F_{j,i}$ should be reset to zero.
- **Rule 3**. Maintain the failure request counting Matrix $F_{n\times m}$, when device j fails to call interface i, $F_{j,i}$ adds 1. If the updated $F_{j,i}$ satisfies $F_{j,i} \geq \mu V_{j,i}$, then $W_{j,i} = \max\left(\lfloor \alpha W_{j,i} \rfloor, 1\right)$ and $S_{j,i}$ and $F_{j,i}$ should be reset to zero.

The key to the "competitive counting" rule is that the variable first reach the threshold in $S_{j,i}$ and $F_{j,i}$ will lead to the change of $W_{j,i}$, and then $S_{j,i}$ and $F_{j,i}$ will be cleared. Therefore, the impact of successful or unsuccessful requests on $S_{j,i}$ and $F_{j,i}$ will not last. $S_{j,i}$ and $F_{j,i}$ can more accurately reflect the group state in a "short time". The parameters that need to be determined are α, λ, μ. The range of them is $0 < \alpha, \lambda, \mu < 1$. An adaptive scheduling algorithm updating $W_{j,i}$ by using the "competitive counting" rule can be uniquely represented by a triple (α, λ, μ).

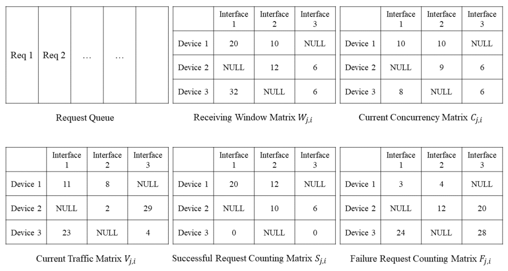
V. Detailed Design of Algorithm

In this section, we describe the detailed design of the adaptive scheduling algorithm.

A. Data Structure Used by The Algorithm

The data structures used by the adaptive scheduling algorithm are mainly queues and hash tables (see Figure 7).

Queues are used to store pending requests. The space complexity is $O(N)$, in which N is the maximum of the queue length, and the average time complexity of the inbound and outbound operations is $O(1)$.



Req 1	Req 2	…	…	

Request Queue

	Interface 1	Interface 2	Interface 3
Device 1	20	10	NULL
Device 2	NULL	12	6
Device 3	32	NULL	6

Receiving Window Matrix $W_{j,i}$

	Interface 1	Interface 2	Interface 3
Device 1	10	10	NULL
Device 2	NULL	9	6
Device 3	8	NULL	6

Current Concurrency Matrix $C_{j,i}$

	Interface 1	Interface 2	Interface 3
Device 1	11	8	NULL
Device 2	NULL	2	29
Device 3	23	NULL	4

Current Traffic Matrix $V_{j,i}$

	Interface 1	Interface 2	Interface 3
Device 1	20	12	NULL
Device 2	NULL	10	6
Device 3	0	NULL	0

Successful Request Counting Matrix $S_{j,i}$

	Interface 1	Interface 2	Interface 3
Device 1	3	4	NULL
Device 2	NULL	12	20
Device 3	24	NULL	28

Failure Request Counting Matrix $F_{j,i}$

Figure 7. Data Structure for Adaptive Scheduling Algorithm

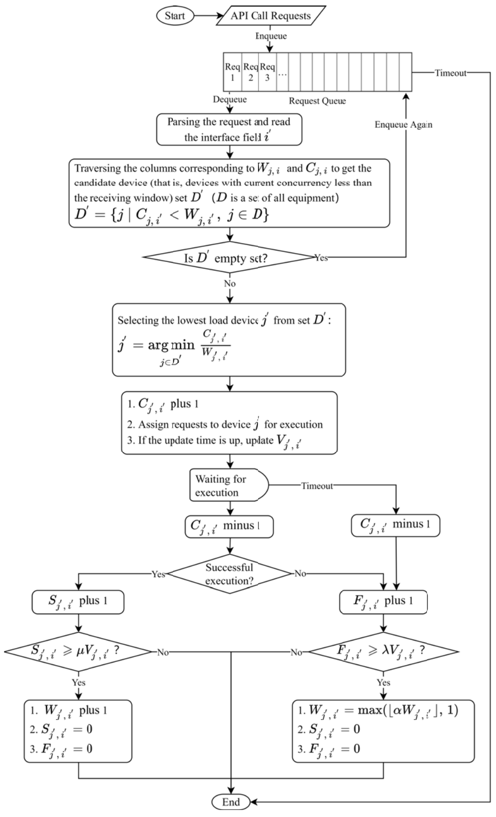

Start
API Call Requests
Enqueue
Req 1
Req 2
Req 3
…
Timeout
Dequeue
Request Queue
Enqueue Again
Parsing the request and read the interface field i'
Traversing the columns corresponding to $W_{j,i}$ and $C_{j,i}$ to get the candidate device (that is, devices with current concurrency less than the receiving window) set D' (D is a set of all equipment)
$D' = \{j \mid C_{j,i'} < W_{j,i'},\ j \in D\}$
Is D' empty set?
Yes
No
Selecting the lowest load device j' from set D':
$j' = \arg\min_{j\in D'} \frac{C_{j',i'}}{W_{j',i'}}$
1. $C_{j',i'}$ plus 1
2. Assign requests to device j' for execution
3. If the update time is up, update $V_{j',i'}$
Waiting for execution
Timeout
$C_{j',i'}$ minus 1
$C_{j',i'}$ minus 1
Successful execution?
Yes
No
$S_{j',i'}$ plus 1
$F_{j',i'}$ plus 1
$S_{j',i'} \geqslant \mu V_{j',i'}$?
No
No
$F_{j',i'} \geqslant \lambda V_{j',i'}$?
Yes
Yes
1. $W_{j',i'}$ plus 1
2. $S_{j',i'} = 0$
3. $F_{j',i'} = 0$
1. $W_{j',i'} = \max(\lfloor \alpha W_{j',i'} \rfloor, 1)$
2. $S_{j',i'} = 0$
3. $F_{j',i'} = 0$
End

Figure 8. Adaptive Scheduling Algorithmic Process

A hash table is used to store $W_{j,i}$ and other global variables. It uses the nested method to access elements according to "device-interface". The element is NULL to indicate there is no corresponding interface deployed on the device. The space complexity is $O(nm)$, in which n is the number of devices, and m is the number of interfaces. The average time complexity of accessing one element is $O(1)$.

B. Algorithm Process

Figure 8 is the detailed process of the adaptive scheduling algorithm. The space complexity of the algorithm is the sum of the space complexity of the global variables, that is, $O(N + nm)$, where N is the maximum queue length, n is the number of devices, and m is the number of interfaces.

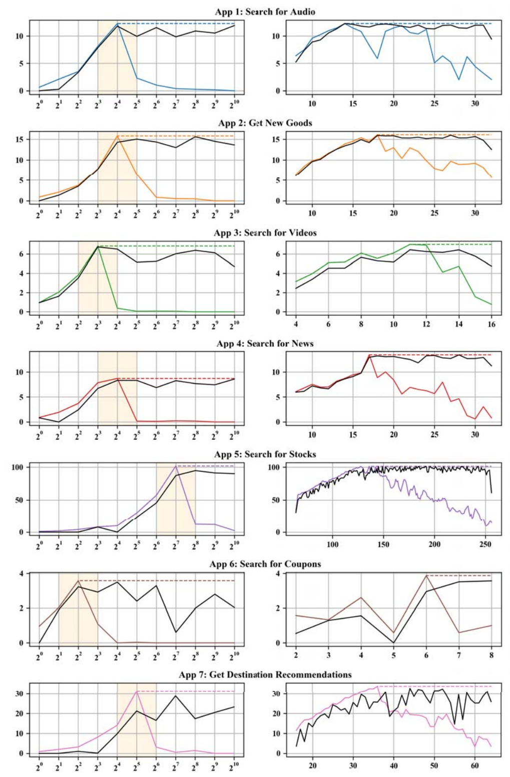


Figure 9. Effect of Adaptive Scheduling Algorithm on Traffic Control (Single Device)

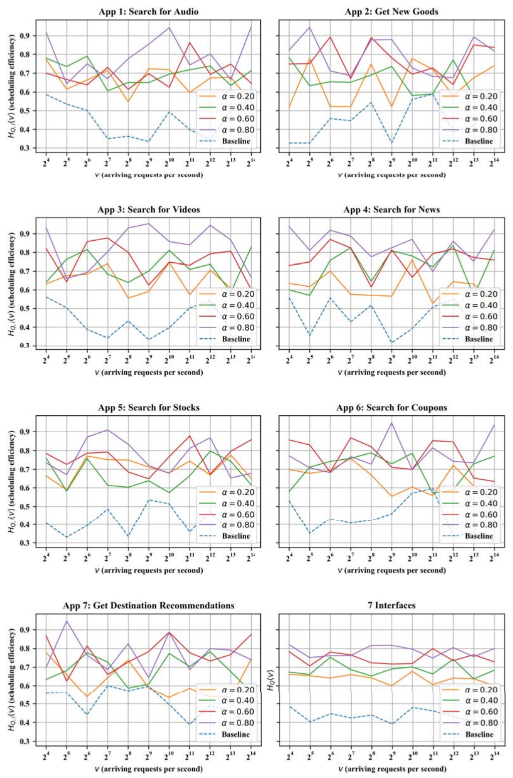


Figure 10. Experimental Results of Parameter μ Optimization

Owing to $N \gg nm$, the space complexity of the algorithm can be approximated to $O(N)$. The time complexity of the algorithm is the sum of the time complexity of the following steps:

- **Step 1**. Calculate the set of candidate devices D': $O(n)$, in which n is the number of devices in the whole group.
- **Step 2**. Choose the target device j': $O(n)$.
- **Step 3**. The time complexity of the other steps is $O(1)$.

Therefore, the time complexity of the algorithm is $O(n)$.

VI. Implementation of Algorithm

The key to the adaptive scheduling algorithm is adaptive traffic control, and the key to the performance of the algorithm is the adjustment rules and parameters of the receiving window. Therefore, there are three steps to implement the adaptive scheduling algorithm:

- **Step 1**. Select the adjustment rules of the receiving window and determine the parameters that need to be optimized.
- **Step 2**. Verify the effectiveness of the algorithm to determine whether the algorithm can effectively control the traffic to the device.
- **Step 3**. Aiming at the specific parameters of the group optimization algorithm, improve the scheduling efficiency.

For **Step 1**, we select the "competition counting" rule introduced in IV.E, and the parameters to be optimized is the triple (α, λ, μ). In this section we mainly introduce **Step 2** and **Step 3**. In VI.A we verify the effectiveness of the algorithm, and in VI.B we optimize the algorithm parameters of the actual group.

A. Algorithm Validity Verification

In this subsection, we verify that the adaptive scheduling algorithm can effectively control the request traffic to the devices. Steps of the experiment are as follows:

- **Step 1**. Build an experimental group consisting of only one gateway server and one working equipment. The configuration of software and hardware of the working equipment is the same as that of III.B.
- **Step 2**. Deploy the adaptive scheduling algorithm on the gateway server, and take any legal value for the algorithm parameter triple (α, λ, μ). Here the parameter is $(\alpha, \lambda, \mu) = (0.5, 0.5, 0.5)$.
- **Step 3**. Apply the method described in III.B, and measure the service capability function $F_i(v)$ of seven behaviour reflective interfaces on the working equipment in the same network environment as III.B.
- **Step 4**. Compare the results with those in Figure 3 and conduct an analysis.

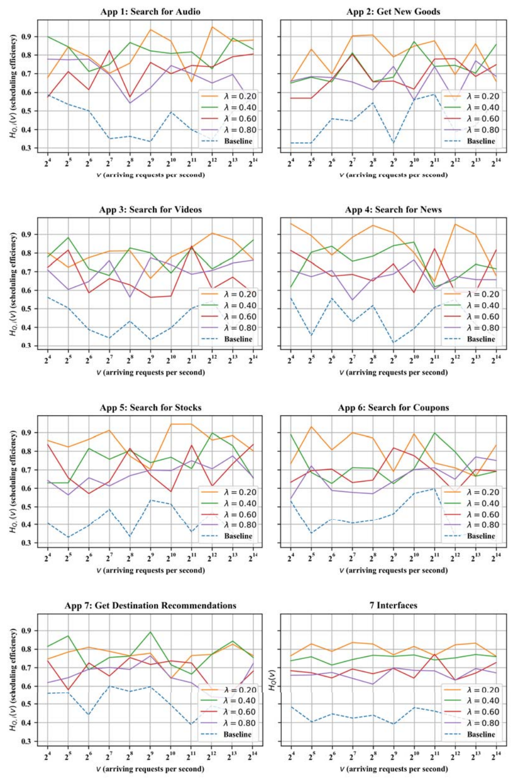


Figure 11. Experimental Results of Parameter λ Optimization

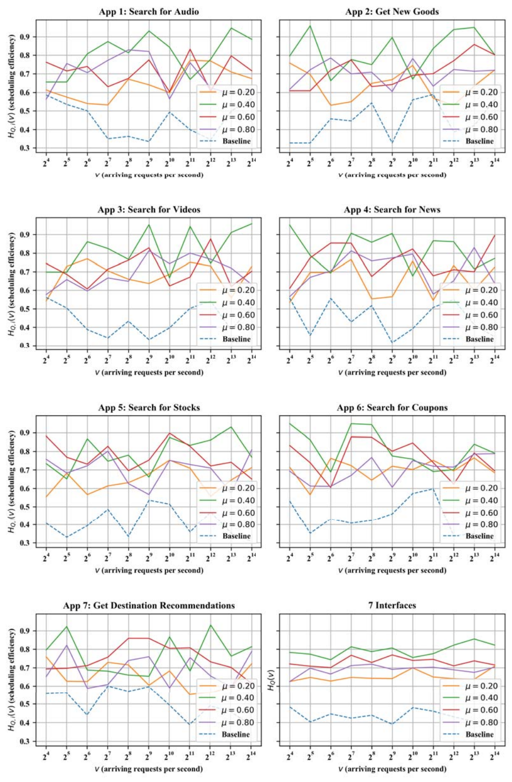


Figure 12. Experimental Results of Parameter α Optimization

The experimental results are shown in Figure 9. The abscissa represents the request arrival rate v, and the ordinate represents the interface service capability $F_i(v)$. The solid black line represents the result of applying the adaptive scheduling algorithm to control the traffic (i.e. the measurement result in this subsection), the solid colour line represents the result of not applying the adaptive scheduling algorithm (see Figure 3). The dotted line represents the ideal value $F_i'(v)$. The request arrival rate v increases exponentially, and then traverses the interval of the critical value v_i^* linearly (see III.D). The results show that:

- When v is less than or equal to the critical value v_i^*, the application of adaptive scheduling algorithm will lead to a small reduction of $F_i(v)$, which may be due to the increased scheduling overhead.
- When v is greater than the critical value v_i^*, the $F_i(v)$ using adaptive scheduling algorithm is significantly higher than that without adaptive scheduling algorithm, and the former is close to the ideal value $F_i'(v)$.

The conclusion is that the adaptive scheduling algorithm can effectively control the request traffic to the devices

B. Algorithmic Parameter Optimization

In this subsection, the parameter triples (α, λ, μ) of the adaptive scheduling algorithm are optimized on the experimental group (16 working devices in total and the behaviour reflection interface of the configuration and deployment of software and hardware is the same as that of III.C). The optimization objective is to maximize the comprehensive scheduling efficiency function of the group $H_Q(v)$ (see IV.B). In the rest of this subsection, we describe the experimental results:

- **Fixed α and λ to Optimize μ.** Let $\alpha = 0.5$,$\lambda = 0.5$, and $\mu \in \{0.2, 0.4, 0.6, 0.8\}$. The experimental results are shown in Figure 10. The "baseline" in the graph is the result of Round-Robin [16] scheduling algorithm. The result shows the best candidate value of Mu is 0.4, so $\mu' = 0.4$.
- **Fixed α and μ to Optimize λ.** Let $\alpha = 0.5$,$\mu = 0.5$, and $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$. The experimental results are shown in Figure 11. It is known that $\lambda' = 0.2$.
- **Fixed λ and μ to Optimize α.** Let $\lambda = 0.5$,$\mu = 0.5$, $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$. The experimental results are shown in Figure 12. It is known that $\alpha' = 0.8$.
- **Verification of Local Optimality**. The optimal combination of parameters is $(\alpha', \lambda', \mu') = (0.8, 0.2, 0.4)$. The experimental results are shown in Figure 13 by comparing the adjacent 26 groups of (α, λ, μ) with $(\alpha', \lambda', \mu')$. It can be seen that $(\alpha', \lambda', \mu')$ is better than 26 groups of (α, λ, μ), so it satisfies the local optimality.

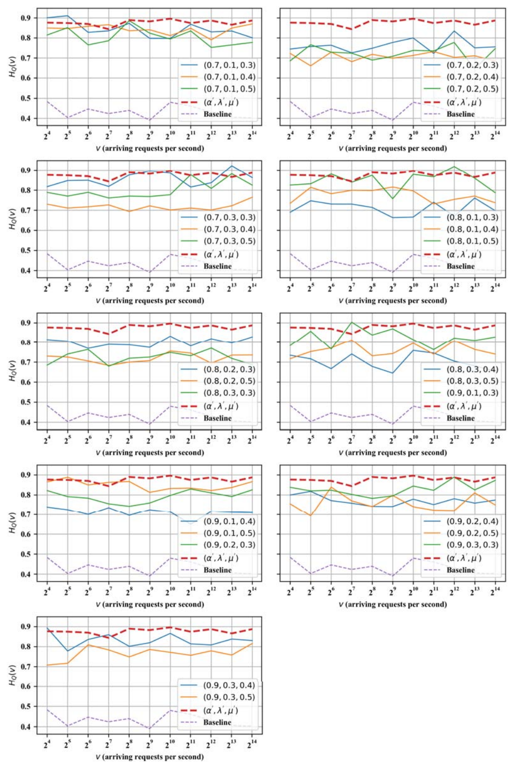


Figure 13. Experimental Results of Local Optimality Verification

VII. Conclusion

This paper mainly completed the following works:

- We define the service capability of a single device in the device cloud and design a method to measure the service capability of a single device.
- We establish an adaptive scheduling algorithm model. We define the overall scheduling efficiency of a device group and determine the maximization of the overall scheduling efficiency as the scheduling goal.
- We verify the effectiveness of the adaptive scheduling algorithm in controlling the traffic on a single device through experiments. We optimize the parameters of the algorithm for a specific group and verify the local optimality of the parameter combination.

In a word, in this paper, we design and implement an adaptive scheduling algorithm for the cloud testing platforms to more efficiently provide services to their users. The effectiveness of the algorithm has been preliminarily verified in the experimental environment and the real scene.

Although the experimental results meet our expectations and research objectives, the algorithm in this paper still has the following limitations:

- **Limitations of parameter assumptions**. In VI.B, we assume that the parameters of the algorithm can be optimized independently, but it can not be proved. Although the results of the experiments are encouraging in the experimental environment of this paper, it can not be guaranteed to be suitable for other situations.
- **Restrictions on smart device groups**. In IV.A, we have made a severe restriction to the device group to which the algorithm applies. The smart device groups of real cloud test platform can hardly meet all these controls, so it is necessary to reduce the restrictions and conduct more experiments.

In the future, we can also carry out the following research based on the research and experimental results of this paper:

- **Other adjustment rules of the receiving window**. In this paper, we put forward the "competition counting" rule, which is simple and effective. However, the "competition counting" rule only utilizes the success request count, failure request count, and current request traffic information. If other tuning rules can take advantage of more information, such as response time, they may get better performance.
- **Different Android versions of smart devices**. The Android testing environment used in this paper is Android 6.0. Although about 12.0% of all Android users use Android 6.0 or 6.0.1 as of April 2020, it is a relatively old version. By testing different Android operating system versions, we can analyze whether there is a relationship between the operating system version and algorithm performance.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under grant number 61725201, China Postdoctoral Science Foundation, and an Open Project Fund of National Engineering Lab of Big Data System Software of China.

References

[1] Techcrunch, "Number of smartphone users," https://techcrunch.com/2015/06/02/6-1bsmartphone-users-globally-by-2020-overtaking-basic-fixed-phone-subscription, 2015.

[2] S. Saxon, J. Neubeck, M. Collins, D. Weinmann, and W. Dorman, "Cloud based test environment," Oct. 6 2011, uS Patent App. 12/899,085.

[3] R. Ghazizadeh and P. Fan, "Dynamic priority scheduling mechanism through adaptive interframe space," Shanghai, China, 2007, pp. 1992 – 1995. [Online]. Available: http://dx.doi.org/10.1109/WICOM.2007.498

[4] Z. Li and M.-Y. Chow, "Adaptive multiple sampling rate scheduling of real-time networked supervisory control system - part i," Paris, France, 2006, pp. 4604 – 4609. [Online]. Available: http://dx.doi.org/10.1109/IECON.2006.347754

[5] Y. Sadi and S. Coleri Ergen, "Energy and delay constrained maximum adaptive schedule for wireless networked control systems," *IEEE Transactions on Wireless Communications*, vol. 14, no. 7, pp. 3738 – 3751, 2015. [Online]. Available: http://dx.doi.org/10.1109/TWC.2015.2411602

[6] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[7] I. Keslassy, C.-S. Chang, N. McKeown, and D.-S. Lee, “Optimal load-balancing,” in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3. IEEE, 2005, pp. 1712–1722.

[8] F. Yang, H. Mei, J. Lv, and Z. Jin, “On the development of software technology,” *Acta Electronica Sinica*, vol. 30, no. S1, pp. 1901–1906, 2002.

[9] G. Huang, Q. Wang, H. Mei, and F. Yang, “Research on reflective middleware based on software architecture,” *Journal of Software*, vol. 14, no. 11, pp. 1819–1826, 2003.

[10] H. Cai, “Research on behavioral reflection of internetware,” Ph.D. dissertation, Peking University, 2017.

[11] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007, no. 7.

[12] T. L. Casavant and J. G. Kuhl, “A taxonomy of scheduling in general-purpose distributed computing systems,” *IEEE Transactions on software engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[13] N. Wells, “Busybox: A swiss army knife for linux,” *Linux Journal*, vol. 2000, no. 78es, p. 10, 2000.

[14] X. Developers, “Xposed framework hub,” 2019.

[15] D.-M. Chiu and R. Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[16] R. V. Rasmussen and M. A. Trick, “Round robin scheduling–a survey,” *European Journal of Operational Research*, vol. 188, no. 3, pp. 617–636, 2008.