Current Journal of Applied Science and Technology

24(6): 1-11, 2017; Article no.CJAST.37852
Previously known as British Journal of Applied Science & Technology
ISSN: 2231-0843, NLM ID: 101664541

Research of SLA-Based Multitask-User-Requests Admission Control and Related Algorithm for the Cloud Service Provider

Zhi-Hong Liang[1], Dong Wang[2*], Fei Dai[1] and Yu-Xiang Huang[2]

[1]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming 650500, China.
[2]School of Software, Yunnan University, Kunming 650091, China.

Authors' contributions

This work was carried out in collaboration between all authors. Author ZHL designed the study, performed the statistical analysis, wrote the protocol and wrote the first draft of the manuscript. Authors DW and FD managed the analyses of the study. Author YXH managed the literature searches. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/CJAST/2017/37852
Editor(s):
(1) Samir Kumar Bandyopadhyay, Professor, Department of Computer Science and Engineering, University of Calcutta, India.
Reviewers:
(1) Borislav Kolaric, Serbia.
(2) K. Gokulnath, Parul University, India.
(3) R. Gomathi, Bannari Amman Institute of Technology, India.
Complete Peer review History: http://www.sciencedomain.org/review-history/22183

Short Research Article

Received 30th October 2017
Accepted 1st December 2017
Published 7th December 2017

ABSTRACT

With the population of Cloud service users showing an explosive growth, how to reduce costs and improve resource utilization while ensuring service quality has become a very important bargaining chip for the cloud service provider can survive in the fierce competition in the industry or not. Based on the service level agreement (SLA) constraints of Cloud service, this paper studies the request admission control strategy of multitask-user-requests. By discussing and analyzing the possible problems in the different stages of request admission, resource scheduling and subtasks execution for multitask-user-requests, tow priority dynamic configuration strategies and a delay compensation strategy are proposed to improve the ratio of requests acceptance, cut down SLA default rates during subtasks are executed, and improve users' experience while the SLA violate occurs. What's more, two resource scheduling strategies aimed at improving the resource utilization rate to reduce

**Corresponding author: E-mail: 1814078959@qq.com;*

the cost of Cloud services as much as possible are proposed. And at the end of the article put forward the Dynamic configuring Priority Resources Scheduling (DPS) algorithm which based on the strategies above, and designed the relevant comparative experiments to verify the algorithm of DPS. The results showed that the DPS algorithm can reduce the usage of resources by improving resource utilization rate and helps the service providers to create much more profits than the other two algorithms to a certain extent.

Keywords: Cloud computing; SLA; admission control strategy; resource scheduling algorithm.

1. INTRODUCTION

With the network bandwidth and the user demand for service quality continues to improve, as a new computing model of cloud computing services are also in line with the trend of the times. Whether it is in the data processing efficiency, or data volume, but also continue to improve and improve [1]. However, due to the complexity of the user's request in the cloud environment, the randomness and the heterogeneity and diversity of the resources under the cloud platform, the scheduling and allocation of cloud resources has always been a NP hard problem. How to accept users' requests reasonably and allocate resources to ensure service providers maximizes profit without violating the SLA constraints of users' requests while guaranteeing the quality of service and enhancing user satisfaction has always been one of the hot spots and difficulties of the research about the cloud resource allocation schedule.

In the literature [2], it was pointed out that the core of the problem of cloud resource scheduling is task scheduling and resource allocation, and the two algorithms are compared with the task queuing model in the time of task execution. However, the author does not take into account the task scheduling and resource allocation required by multitasking users, and does not incorporate user request SLA constraints into the cloud service impact factors. In the literature [3], the cloud service provider could not solve the problem of virtual machine rental cost through the effective resource scheduling strategy under the premise of guaranteeing the user SLA when dealing with the workflow application request in the cloud environment. The author proposes a multi-workflow virtual machine resource scheduling algorithm, which effectively reduces the cost of virtual machine. However, this profit-driven cloud resource scheduling algorithm does not handle interactive application requests very well. In the literature [4], the profit model under the two roles was analyzed and defined from the perspective of IaaS suppliers and SaaS suppliers. Then the constraints of user requests based on SLA, the four resource scheduling strategies were integrated into three resource scheduling algorithms, so as to reduce the resource usage and maximize the profitability of the service provider as much as possible. Although the experiment achieves the expected goal, it only considers the case that the user request only contains a single task, and does not consider the request of the multi task user. In the literature [5], a fine-grained cloud computing system model was introduced, using reinforcement learning and queuing theory to optimize the resource constrained scheduling strategy based on, and through the state to strengthen the convergence of learning progress in order to improve the accuracy of the cloud platform user request analysis. Although it is emphasized that the SLA constraint requested by the user is one of the influence factors of the cloud service, it only considers the single subtask in the user request.

In view of the above problems, this paper studies the admission control strategy of multi-tasking users based on the SLA constraint from the perspective of cloud service agents. The profit model is analyzed and formalized. Finally, a priority resource allocation resource scheduling algorithm is proposed to achieve the goal of maximizing the profitability of cloud service agents by maximizing resource utilization.

2. MODELING OF MULTI TASK USER REQUEST AND PROFIT MODEL BASED ON SLA IN CLOUD ENVIRONMENT

2.1 Multitasking User Request Modeling

In the cloud services three-tier architecture, IaaS vendors through virtualization technology, different hardware facilities virtualized into a large-scale resource pool, to cloud service agents to provide computer capacity, network, storage and other infrastructure services. Cloud service agents build a variety of PaaS and SaaS

service platforms by leasing their infrastructure to provide the end user with the required PaaS and SaaS services. Finally, users pay cloud services to cloud service agents on a pay-as-you-go basis.

In these user requests, a user request often contains not only a single task, but by a series of parallel or serial multiple sub-tasks staggered in series, and the sub-tasks between the existence of strict precursor successor, as shown in Fig. 2.

This paper presents the multitasking user request as a multivariate set expression as follows:

$$Q = \{T, E, W, C\}.$$

Among them, the set T is cumulative task in a user request, and T_i represents the subtask *i* in the user request. The set *E* is a set of directed edges, representing the precursor dependency between the subtasks, and E_{ij} denotes that the subtask *j* depends on the subtask *i*. The set *W* is the set of processing time for the current subtask, and the processing time of the subtask *i* is W_i . The set *C* represents the length of information transfer between the subtasks, and C_{ij} represents the length of time that the execution result of the subtask *i* is passed to the subtask *j*.

When a user requests that a parent task has a predecessor task in the process of being executed, all predecessor tasks must be completed to execute the subtask. However, during the execution of the subtask, its successor task can only be in a wait state. After the current subtask is executed, the execution result is immediately passed to its subsequent task.

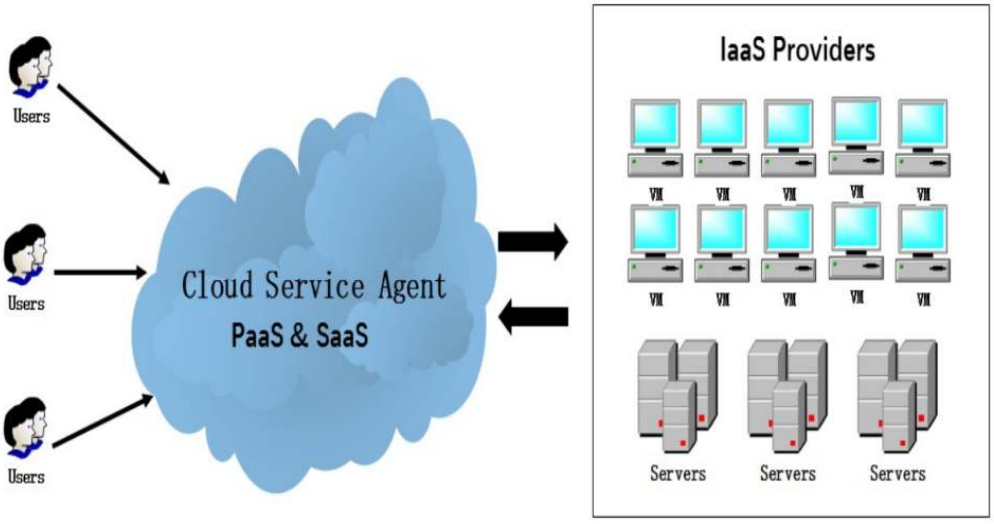

Fig. 1. Cloud service three-tier architecture

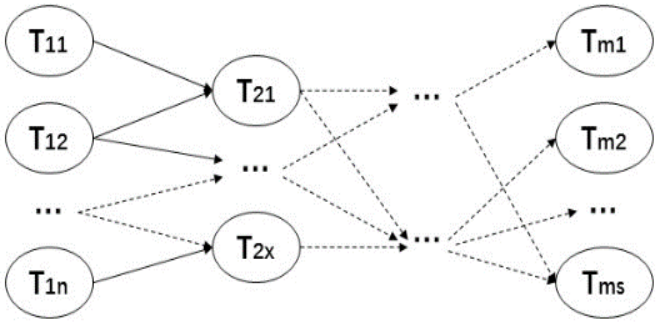

Fig. 2. User request neutron task dependency diagram

2.2 Profit Model Analysis and Formal Definition of Cloud Service Agents

Cloud service agents rent infrastructure to IaaS providers to build their own cloud services platform, and then provide cloud services to end users. So we can find that the revenue of the cloud service agent is the cost of the end user. At the same time, the cost of cloud service agents is divided into three parts: platform deployment costs ($Cost_{Deploy}$)、IaaS service rental costs ($Cost_{Rent}$)、platform management related costs ($Cost_{Manage}$). Among them, the platform management related costs are relatively stable, we set it as a fixed cost.

The cloud service agent requests the I type of virtual machine resource *i* on the IaaS provider j to deploy the cloud service platform for the cost of $DCost_{ijl}$. With A IaaS service provider leasing, each IaaS supplier leases Class B services, each type of service has virtual machine resources C, the total deployment cost is:

$$Cost_{Deploy} = \sum_{j=1}^{A}\sum_{l=1}^{B}\sum_{i=1}^{C}(DCost_{ijl}) \quad (1)$$

Since IaaS services are generally charged on time, the paper makes the following assumptions:

The IaaS service has a unit time charge of m_P, and a user request has *N* subtasks, each of which has a processing duration of $m\_ProcT(i)$ $(0 < i \le N)$. The data input time and data output time for each subtask is DI_T_i and DO_T_i. The cost of data input and data output is In_P/GB and Out_P/GB, and the initialization time of each virtual machine is $\mathrm{In}it\_T$. For the delay of user service, the linear compensation strategy is adopted, and the compensation factor is $\beta(0 < \beta < 1)$. Then, in the case of multitasking user requests, the rental fees for the cloud service agents are:

(1) The execution cost of all subtasks in the user request:

$$ProcCost = \sum_{i=1}^{k}(m\_ProcT(i) \times m\_P) \quad (2)$$

Where k is the number of subtasks on the critical path that the user requests to perform during the execution of the user request.

(2) The user requests the total data transmission cost:

$$DT\_Cost = \sum_{i=1}^{N}(DI\_T_i \times In\_P + DO\_T_i \times Out\_P) \quad (3)$$

(3) The cost of initializing a virtual machine:

$$initVM\_Cost = (Init\_T \times m\_P) \quad (4)$$

(4) User compensation costs:

For a new user request, the admission control system used in two ways: (a) initialize $W(0 < W \le N)$ VM resources for the requested N subtasks. (b) add subtasks in the user request to the initialized VM task list. Assuming that there are *M* pending tasks in the task list, the processing time of each task to be executed is $ProcT_m, (0 < m \le M)$, the maximum time required for the user is T_{DL}, the user sends a request for T_u. The information transfer time between tasks is T_{Tr}, *N* sub-tasks have at least *(N-1)* times information transmission, and the delay time of the user request is:

$$T_{Delay} = \begin{cases} T_U + \sum_{m=1}^{M} ProcT_m + \sum_{i=1}^{N} m\_ProcT(i) + \sum_{i=1}^{N-1} T_{Tr} - T_{DL}, (Strategy\ b) \\ T_U + \sum_{i=1}^{N} m\_ProcT(i) + \sum_{i=1}^{N-1} T_{Tr} + Init\_T \times W - T_{DL}, (Strategy\ a) \end{cases} \quad (5)$$

It can further calculate the user delay compensation cost as:

$$Delay\_Cost = T_{Delay} \times \beta(0 < \beta < 1) \quad (6)$$

Where β is the compensation factor.

(5) The transfer cost of information during user request execution:

$$InfoTr\_Cost = \sum_{i=1}^{N-1} T_{Tr} \times m_P \quad (7)$$

From the above analysis, cloud service agents to release IaaS service costs:

$$\mathrm{m\_Cost_{Total}} = \mathrm{ProcCost} + \mathrm{DT\_Cost} + \mathrm{initVM\_Cost} + \mathrm{Delay\_Cost} + \mathrm{InfoTr\_Cost} \quad (8)$$

The paper assume that the user budget is B_U, then the profit model for the cloud service agent is:

$$m_{Profit_A} = B_U - m_{Cost_{Total}} - Cost_{Deploy} - FCost_0 \quad (9)$$

3. SLA-BASED MULTITASKING USER REQUEST ADMISSION CONTROL STRATEGY AND RESOURCE SCHEDULING ALGORITHM

3.1 SLA-based Multitasking User Request Admission Control Strategy

In order to solve the admission control problem of user request and the SLA default in the process of user request execution. In this paper, user request priority setting strategy, task execution priority setting policy and request delay compensation strategy are proposed respectively. Through the combination of these three strategies, as much as possible to accept user requests, improve customer satisfaction, cloud service agents to maximize the profits to create profits.

The paper first assumed that there is no waiting time between the sub-tasks in the user's request, and that each sub-task is executed immediately after the allocation of resources. This moment it was call the ideal situation. Because a sub task takes up resources, the length of time is divided into two cases: (1) if the task is assigned to the initialized virtual machine, there are data input, output time, task processing time and information transfer time. (2) if the new virtual machine is initialized for the task, there are data input, output time, task processing time, Thus, the resulting sub-task occupancy resource time model is:

$$T_{Proc} = \begin{cases} DI\_T_i + DO\_T_i + m\_ProcT + T_{Tr} & (1) \\ DI\_T_i + DO\_T_i + m\_ProcT + T_{Tr} + Init\_T & (2) \end{cases} \quad (10)$$

Then, when a user request has N subtasks and allocates resources in accordance with strategy a: *W* virtual machine is initialized and the user requests the execution time of the neutron task. The total execution time on the critical path is set to T_{Max}. There are:

$$T_{Max} = \sum_{i=1}^{X} m\_ProcT(i) \quad (11)$$

Where *X* is the number of subtasks on the critical path of the task execution time. Then the paper can draw the expected processing time model for the user's request under the ideal situation:

$$T_{RP} = \begin{cases} T_U + T_{Max} + \sum_{i=1}^{N-1} T_{Tr} + \sum_{i=1}^{N} (DI\_T_i + DO\_T_i), & (Strategy\ b) \\ T_U + T_{Max} + \sum_{i=1}^{N-1} T_{Tr} + Init\_T \times W + \sum_{i=1}^{N} (DI_{T_i} + DO_{T_i}), & (Strategy\ a) \end{cases} \quad (12)$$

But the ideal is always a gap with reality, so the paper find closer to the actual situation of the user request is expected to deal with time:

$$T_{Acc} = T_{RP} + \frac{\sum_{i=1}^{M} P_{URP}(i)}{M \times P_{URP}} \times \frac{\sum_{i=1}^{N} \mathrm{m\_ProcT(i)}}{N} \quad (13)$$

Among them, *M* is the number of user requests in the current unit time, and $P_{URP}(i)$ is the priority of the *i* user requests in *M* user requests.

3.1.1 The user requests the priority setting strategy

Fig. 3 is a model that dynamically sets a priority for a user request based on a user request SLA time limit constraint and a user request processing time. When the user's time limit is equal to the processing time of the user request, the highest priority is set for *P_0*. As the user time limit is gradually greater than the user requests the expected processing time, the user request priority drops linearly. The linear model is:

$$P_{URP} = P_0 - (T_{DL} - T_{Acc}) \times \tan\theta, \quad \left(T_{DL} \geq T_{Acc},\ 0 < \theta < \frac{\pi}{2}\right) \tag{14}$$

Where *tanθ* sets the coefficient for the user request priority. When a user's request is accepted by the system, its subtask has the same priority initial value as the user request.

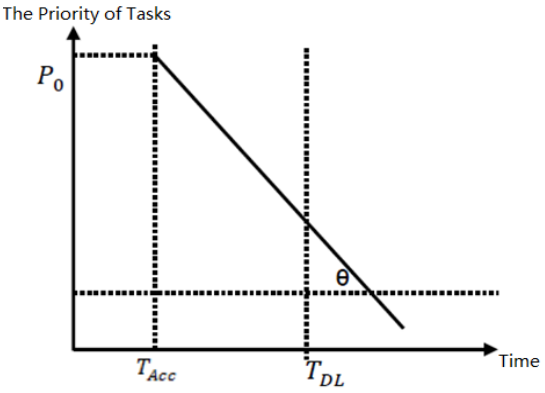

Fig. 3. The user requests the priority setting model

3.1.2 Task execution priority setting strategy

During the execution of the sub-tasks requested by the user, the randomness of the resource allocation and the uncertainty of the user's request often result in some sub-tasks being unable to be completed within the SLA constraint time, resulting in serious default user request SLA. To this end, the task execution priority setting policy is presented, as shown in Fig. 4.

When a task fails to be executed within its constraint time, the task's execution priority increases linearly with time until the task is executed. Its linear model is:

$$P_{Exc} = P_0 + tan\,\beta \times (T_{End} - T_{DL}),\ \ 0 < \beta < \frac{\pi}{2} \tag{15}$$

Where T_{End} is the time at which the task is finally executed, and T_{DL} is the SLA time limit constraint for the task. P_0 is the initial priority of the task. *tanβ* sets the coefficient for the task execution priority.

3.1.3 Request delay compensation strategy

In order to reduce the impact of the default on the user experience when the user requests the SLA default, the compensation method may be used to improve the user satisfaction. As shown in Fig. 5, the user requests the delay compensation model.

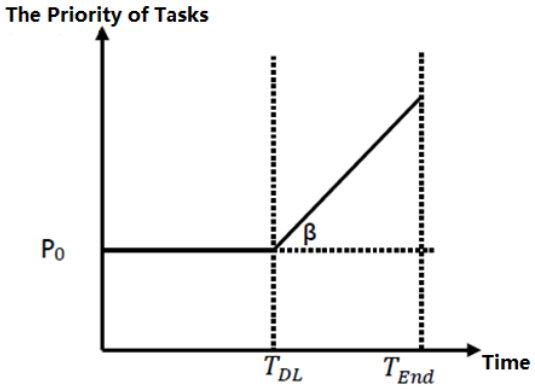

Fig. 4. Task execution priority setting model

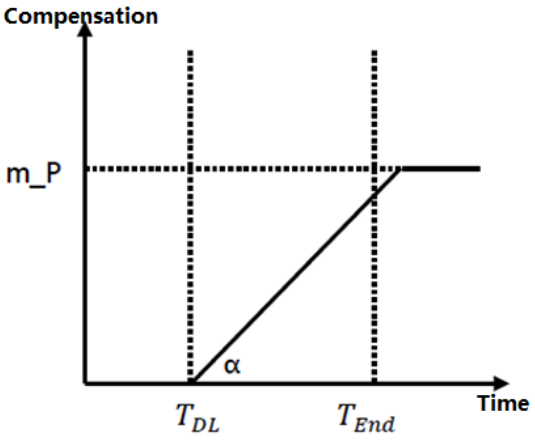

Fig. 5. Delayed compensation model

When the user requests a SLA default, the cloud service agent compensates the user. The compensation model is:

$$Pric_{Com} = (T_{End} - T_{DL}) \times \tan\alpha, (Pric_{Com} \leq m\_P, T_{End} \geq T_{DL},\ \ 0 < \alpha < \frac{\pi}{2}) \tag{16}$$

Among them, T_{End} requests execution end time for the user, and tanα is the delay compensation factor. *m_P* charges for cloud resource providers per unit time, and is also the maximum limit for cloud service agents to provide compensation to users.

3.2 Resource Scheduling Algorithm Based on SLA

After a user request is accepted by the system, the following two strategies are used for resource scheduling:

3.2.1 Waiting strategy

When a multitasking user requests are accepted by the system:

1. Determine the SLA constraint for the task.
2. Check the task queue in the system that has been initialized and satisfies the task requirements for virtual machine resources. If the task can wait for the task queue of the virtual machine to complete without violating the SLA constraint, the next step is executed. Otherwise exit the strategy.
3. Calculate the profit earned by cloud service agents. If the profit is greater than the expected value, the task and resource allocation information is recorded in the resource-task list. Otherwise exit the strategy.

3.2.2 Insert strategy

When a multitasking user requests are accepted by the system:

1. Determine the SLA constraint for the task.
2. Check whether task K exists in the recently initialized virtual machine resource task queue. Without violating the SLA constraints under the premise, you can wait for the implementation of the current task is completed. If it exists, proceed to the next step. Otherwise exit the strategy.
3. Set the execution priority higher than task *k* for the current task.
4. Calculate the profit earned by cloud service agents. If the profit is greater than the expected value, the task and resource allocation information is recorded in the resource-task list. Otherwise exit the strategy.

Based on the above two resource scheduling strategies, this paper proposes a dynamic configuration priority resource scheduling algorithm based on SLA. The algorithm is divided into four phases: the user requests the admission control phase, the task resource allocation phase, the resource scheduling phase and the task execution phase.

Dynamic allocation priority resource scheduling algorithm based on SLA (DPS)

Input: NR, EP *// NR indicates a new request and EP indicates the expected profit of Cloud service provider*
Output: Blooean
BEGIN
 # Phase of admission control
1: *T_Acc* ← calculate the expected execution time of NR
2: if *T_Acc* ≥ *T_DL*
3: reject NR
4: else
5: P_0 ← set the priority of NR
6: check resource pool
7: Res_avail ← available resources that fulfill the user's request for execution
8: if Res_avail not exit
9: reject NR
10: else
11: accept NR
 # Phase of resource allocation
12: for subtask in NR
13: if NR can be executed by *waiting strategy*
14: follow *waiting strategy* to assign resource to NR
15: go to step 20
16: else if NR can be executed by Insert strategy

17: follow *Insert strategy* to assign resource to NR
18: go to step 20
19: else initialize new vm for NR
Phase of resource scheduling
20: Res_cor ← get corresponding resource from the resource-tasks list
21: schedule NR to Res_cor to be processed
Phase of task execution
22: Subtasks_vio← subtasks that has violated the SLA constraints
23: for Subtask_v in Subtasks_vio
24: $P_t \leftarrow P_{Exc}$ of Subtask_v
25: end for
26: tasks are executed according to the level of tasks priority P_t
27: if NR violates the SLA constraints of user's request
28: compensate user according to *request delay compensation strategy*
29: end for
30: return the result of NR
31: end if
32: end if
END

4. EXPERIMENTAL PRESENTATION AND ANALYSIS

4.1 Experimental Design

4.1.1 Design of experiment scheme

In this paper, Workflow Generator tools are used to simulate *N* subtasks. There are seven sub-tasks in each user request, the processing time of each subtask, the dependencies between subtasks, and the time consumption between subtasks, as shown in Fig. 6.

Where the set *T* represents the cumulative task node, and the number above the node represents the processing time of the cumulative task. The directed edge represents the dependency between the subtasks, and the weight on the directed edge represents the time consumption between the cumulative task.

In addition, in order to simplify the experiment process, the following regulations are made:

(1) The user request must be generated in accordance with the Poisson distribution. The total number of user requests is 200, 400, 600, 800, 1000 five groups to experiment.
(2) The maximum virtual machine resource is 100.
(3) The implementation constraint for each user request is: $T_{DL} = MaxT \times (1 + \alpha)$. Among them, α=0.3 is a variable constraint factor, and *MaxT* is the task execution time of the critical path between user requests and neutron task execution.
(4) Cloud services are charged when resources are used. IaaS vendors charge are *P_i*=0.3. The service default probability is 0.25. User time bound factor is *α*=0.3. Service compensation factor is *β*=0.2.
(5) The user budget is *B_u*=30. Platform integration costs and platform management costs are not considered.
(6) For other parameter settings during the experiment, it is required to obey Gaussian distribution.

4.1.2 Reference algorithm design

To avoid the contingency in the experiment, two reference algorithms are listed here:

(1) BS (Backfill Scheduling) algorithm [6]

The algorithm is based on the FCFS (First Come First Serve) scheduling algorithm, which allows the job in the job queue to be executed in advance using idle resources without delaying the execution of the front end of the queue. In this way, it can solve the problem of idle computing resources and low resource utilization in the FCFS scheduling process. The fundamental purpose of the algorithm is to run the smaller jobs as early as possible. In improving the utilization of system resources and task response time, while avoiding some of the larger jobs were "starved to death" [7].

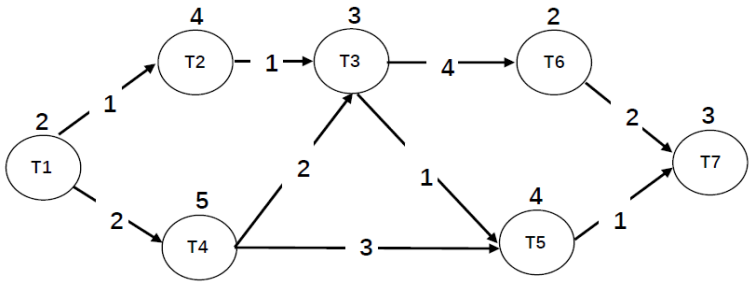

Fig. 6. Multitasking user request

(2) Earliest Deadline First (EDF) algorithm [8]

The algorithm configures the priority according to the time constraint length of the current task. The shorter the task time limit, the higher the priority of the task. The system will assign scheduling resources based on the priority of the task. In the execution of a task, if there is a task higher than the current priority. Immediately stop the execution of the current task and turn to a higher priority task. The execution of the original task is resumed until there is no high priority task in the task queue. The algorithm is a dynamic priority scheduling algorithm, which is proved to be able to achieve dynamic optimal scheduling. Resource utilization can also be up to 100%. However, insufficient is likely to cause system overloads. Once a task is lost, it often causes a series of tasks to be lost [9].

4.2 Experimental Results Show and Analysis

The experiment mainly from the multi-tasking user request under the three algorithms, the virtual machine resource occupancy and profit situation comparison.

4.2.1 Virtual machine resource occupancy analysis

As shown in Fig. 7, under the three algorithms, the virtual machine resource occupancy is increased as the number of user requests increases from 200 to 1000 step by step. Among them, the BS algorithm always maintains a higher amount of resources, because the algorithm mainly meets the increasing user requirements by constantly initializing the virtual machine resources for the new task. The EDF algorithm has the least amount of resource occupation when the user requests less. But when the user requests increase to 800, the occupation of resources is skyrocketing. Because the number of users is small, the algorithm can guarantee higher resource utilization. However, when the user data increases, the system will gradually appear overload situation, so the need to constantly initialize the virtual machine resources to meet the needs of user growth. In the whole process, the DPS algorithm can maintain a more balanced resource occupation no matter whether the number of users is small or more.

4.2.2 Analysis on profit acquisition of cloud service agents

As shown in Fig. 8, with the increase in the number of users, cloud service agents to obtain profits also increased. In the process of increasing the number of user requests from 200 to 400, the EDF algorithm gains the highest profits, and the BS algorithm achieves the least profit. However, with the further increase in the number of user requests, DPS algorithm and BS algorithm to obtain the profits gradually more than EDF algorithm. The DPS algorithm gains less profit than the EDF algorithm when the user requests less. However, in the process of increasing user requests, the algorithm can maintain a relatively stable growth trend.

From the above experimental results, EDF algorithm has a great advantage in terms of resource consumption and profit acquisition when the user requests less. However, with the increase of the number of users, the problem of EDF algorithm is gradually highlighted. From the overall view of the experiment, the DPS algorithm has a better performance in terms of resource occupation and profit gain when dealing with more user requests. Therefore, the DPS algorithm is more suitable for strict compliance with SLA constraints. The algorithm can also achieve the desired goal to a certain extent on the resource scheduling problem of cloud service agents with more user requests. The DPS algorithm uses less resources to create as much profit as possible.

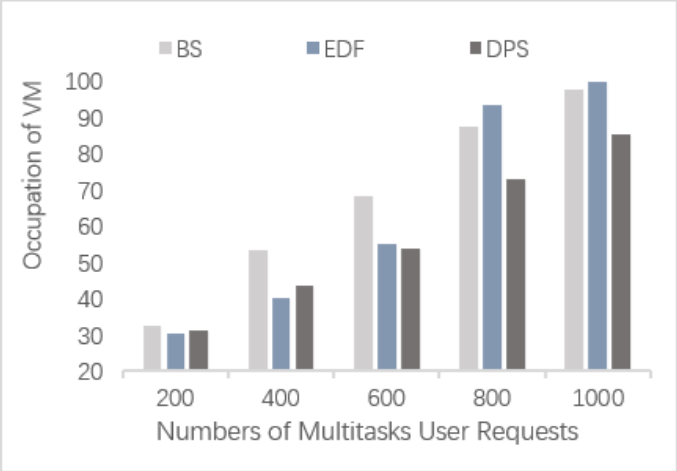

Fig. 7. Resource occupancy under three algorithms

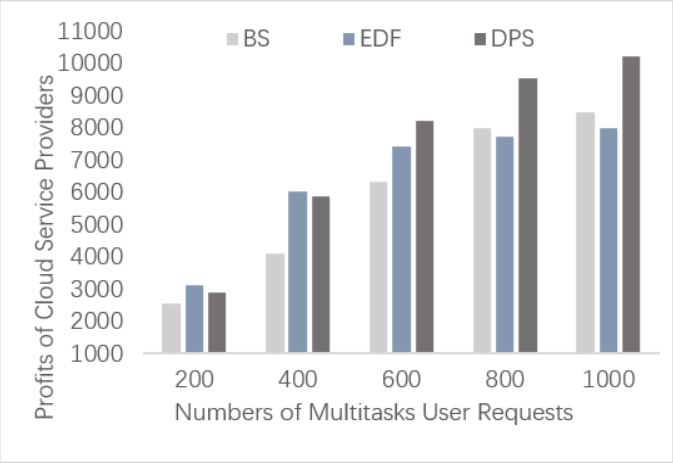

Fig. 8. Profitability of cloud service agents

5. CONCLUSION

This paper analyzed and formalized the profit capture problem and profit model of cloud service agents in the cloud environment. Then, according to the admission control problem requested by the multitasking user, a user request admission control strategy is proposed to accept the user request as much as possible. For SLA defaults that may exist in task execution, task priority setting strategy and user request delay compensation strategy are proposed respectively. This will minimize the SLA default during task execution. As well as in the event of a SLA default, through the compensation way to ensure customer satisfaction. Finally, a dynamic allocation priority resource scheduling algorithm based on SLA is proposed based on two resource scheduling strategies. In addition, the algorithm was verified by experiments. The final experimental results show that the algorithm can generate more profit benefit for cloud service agents by reducing the amount of resource occupation.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Shengjun Xue, Wenling Shi and Xiaolong Xu. A heuristic scheduling algorithm based on PSO in the cloud computing environment. Science and Engineering

Research Support Society [J]. 2016;(9): 349-362.
2. Gawali MB, Shinde SK. Implementation of IDEA, BATS, ARIMA and queuing model for task scheduling in cloud computing [C]. Fifth International Conference on Eco-Friendly Computing and Communication Systems. IEEE. 2017;7-12.
3. Hu R. Research and realization of high benefit resource scheduling mechanism in cloud computing [D]. Beijing: Beijing University of Posts and Telecommunications; 2012.
4. Linlin Wu, Saurabh Kumar Garg, Rajkumar Bu-Yya. SLA-based admission control for a software-as-a-Service Provider in Cloud computing environments [J]. Computer and System Sciences. 2012;78:1280-1299.
5. Peng Z, Cui D, Zuo J, et al. Random task scheduling scheme based on reinforcement learning in cloud computing [J]. Cluster Computing. 2015;18(4):1595-1607.
6. Li W, Shi H. Dynamic load balancing algorithm based on FCFS [C] 2009. 4th International Conference on Innovative Computing, In Formati-on and Control. [s.1.]: [s.n.]. 2009;1528-1531.
7. Fu YH. Research on parallel computing job scheduling algorithm based on backfill [D]. Changsha: Hunan University; 2007.
8. Jiang H. Research on the earliest insertion time of task based on EDF algorithm [D]. Changsha: Hunan Normal University; 2012.
9. Li X J, Li K. A RQ job scheduling algorithm based on dynamic priority [J]. Small Microcomputer System. 2017;38(1):124-128.

Peer-review history:
The peer review history for this paper can be accessed here:
http://sciencedomain.org/review-history/22183