Enhancing Load Balancing in Cloud Computing Through Deadlock Prediction

Hieu Le Ngoc[1(✉)] and Hung Tran Cong[2]

[1] Ho Chi Minh City Open University, Ho Chi Minh City, Vietnam
hieu.ln@ou.edu.vn
[2] Posts and Telecommunication Institute of Technology, Ho Chi Minh City, Vietnam
conghung@ptithcm.edu.vn

Abstract. Cloud computing has become a crucial aspect of Information Technology, offering solutions to many of the challenges faced by internet users. However, the increasing number of users worldwide has led to congestion at certain nodes, resulting in unbalanced loads or hanging systems, commonly known as deadlock. This paper proposes an algorithm using deadlock prediction to enhance the load balancer in Cloud environment. This algorithm leverages Machine Learning and prediction techniques, specifically the Linear Regression Model, to forecast the possibility of deadlock in VMs. By predicting deadlock, the available resources can be allocated to satisfy all requests. The algorithm was deployed in the CloudSim simulation environment, which was integrated with Weka library for the Machine Learning techniques. The results were compared to well-known algorithms such as FCFS, RoundRobin, MaxMin, and MinMin. The evaluation revealed that the proposed algorithm outperformed these popular algorithms, demonstrating its effectiveness in enhancing Cloud computing's load balancing capabilities.

Keywords: Cloud Computing · Load Balancing · Deadlock Prediction

1 Introduction

Cloud computing, according to [1] and [2], is a rapidly growing field in information technology that has the potential to revolutionize the industry. IT developers can now create innovative internet services without needing significant investments in hardware or human resources to operate them. Through applications such as web browsers, mobile apps, and personal computers, many people can access and use cloud services.

Load balancing is crucial for maximizing the benefits of cloud computing. It helps to maintain a balanced state in the cloud and improve services for clients by evenly distributing resources and reducing deadlock. Load balancing algorithms typically focus on performance and economic metrics, scheduling and allocating algorithms, and heuristic or optimization-based enhanced balancers. However, internal factors are not the only ones that affect load balancing in the cloud. External factors, such as the network, users, and geography, can also impact its performance. Deadlock on the cloud can also occur due to user behavior.

N.-S. Vo and H.-A. Tran (Eds.): INISCOM 2023, LNICST 531, pp. 257–274, 2023.
https://doi.org/10.1007/978-3-031-47359-3_19

This paper takes an external perspective on load balancing in the cloud and focuses on the issue of deadlock. The authors propose a new approach for predicting deadlock, with the aim of reducing congestion and hanging status when distributing resources among virtual machines (VMs). To model deadlock, they use linear regression based on historical usage data of the VMs, taking into account both request properties and cloudlet features. Using this approach, the authors aim to allocate requests to the appropriate VM, based on the predicted usage data and the current status of all VMs, in order to avoid deadlock. The propose using thresholds for CPU, RAM, and storage usage to detect when a VM is likely to cause a deadlock and to allocate the request to a different VM. This approach has the potential to improve load balancing in cloud computing.

This article makes several key contributions. Firstly, it approaches load balancing in cloud computing from an external perspective, specifically examining the occurrence of deadlock in VMs. Secondly, it proposes a novel approach for predicting deadlock using Linear Regression Model and applying thresholds. Finally, it presents an experimental evaluation of the proposed algorithm, Deadlock Prediction Algorithm, which outperforms existing algorithms in load balancing.

To make our proposal more accessible, we have structured the paper into 5 sections. The first section, which is the current section, serves as the introduction. The following section reviews and examines related studies. Section 3 presents and explains our proposed algorithm, Deadlock Prediction Algorithm (DPA). In Sect. 4, we discuss the simulation and experiment outcomes, as well as the evaluations. Finally, in Sect. 5, we summarize the main findings of the paper and suggest possible future research.

2 Related Work

Based on references [3] and [4], Cloud Computing is a computing model that utilizes computer technologies and internet-based development. It is also known as virtual server with pay-per-use. Under this model, all resources, software, and information are shared and provided as services to computers, devices, and users over a public network, typically the internet, on an infrastructure platform. Figure 1 gives us an easy imagination about how computing is structured.

Load Balancing [5, 6] is a crucial technique in cloud computing that improves server performance by effectively distributing resources and avoiding deadlocks. It has several key functions, including blocking network traffic to prevent overloading on a single server, distributing loads to servers for processing, handling, and returning results. Load Balancing can also offer redundancy by using multiple failover scenarios.Subsequent paragraphs, however, are indented. Figure 2 illustrates the role of cloud load balancing in the internet.

Deadlock is a computer problem, and it is also a system problem [8–18]. Deadlock is a situation where a group of processes becomes stuck in an infinite waiting state, and this can occur in various computing systems, including cloud, distributed, and grid computing. The advancement of machine learning and data analysis techniques has made it possible to predict the occurrence of deadlock in cloud environments. This prediction can be useful in efficiently allocating resources, particularly virtual machines, on the cloud. Figure 3 shows the unsafe state and safe state and the possibilities of deadlock in allocating resources.

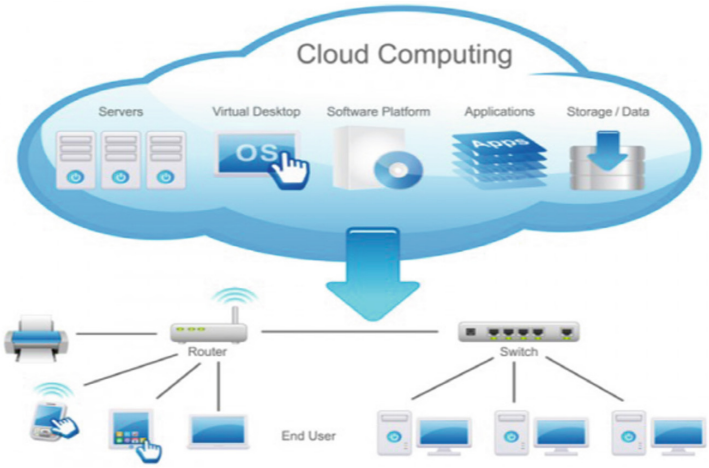

Fig. 1. Cloud Computing model (source: https://informationq.com/)

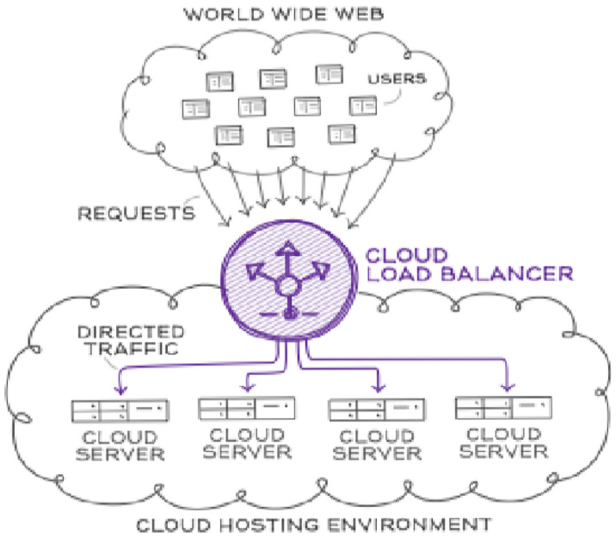

Fig. 2. An example of Cloud load balancer [7]

This article introduces a novel approach in cloud computing, which is predicting the usage of resources for a request. While there have been studies on using ML in distributed systems to predict resource usage, this approach is relatively new. Various ML methods can be used to predict resource usage, each with its own accuracy level [20, 21]. Linear Regression [22], Regression Trees [23], Bagging using Regression Trees [24], and Artificial Neural Networks [25] are some popular ML techniques for resource usage prediction. After evaluating these methods, this paper focuses on using Linear

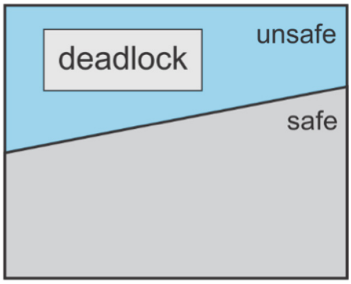

Fig. 3. Safe, unsafe, and deadlock state spaces [19]

Regression on historical data of processing requests from VMs to predict the resource usage of future requests, for the purpose of runtime response.

In 2012, Rashmi K.S and their team [8] from India proposed an algorithm to avoid deadlock in cloud computing, recognizing that a deadlock is possible due to decentralization and virtualization. Their algorithm aimed to increase the amount of processing work in the cloud for better service and to avoid deadlock, analyzing the capabilities and availability of virtual machines and updating the data structure. Although the proposed algorithm was a good approach at the time, it may need to be improved to fit more complicated and scalable cloud systems. In 2013, J.Lim et al. from Korea [9] proposed a scalable and fault-tolerant deadlock detection algorithm for cloud computing using gossip protocol. Their proposed algorithm considered circular wait and violations of safety and liveness properties. They used O(n) to evaluate the algorithm and the PeerSim simulator for the experiment, which showed significant improvement over existing algorithms in solving scalability and fault-tolerance issues. Mahitha.O et al. [10] proposed an efficient load balancing technique in their 2013 article "Deadlock Avoidance through Efficient Load Balancing to Control Disaster in Cloud Environment", to control deadlock issues in Cloud. The authors focused on the deadlock occurring due to a loop inside a VM where a job waits for another resource, and this resource also waits for that job. The proposed tuning algorithm uses CloudAnalyst tool to compare cloud performance with and without a load balancer, resulting in better resource management and improved response time. Figure 4 is the approach of the research by Mahitha.O et al.

In 2015, Cuong Nguyen and his co-author [11] improved the detection and avoidance of deadlock in heterogeneous distributed platforms. The authors highlighted that the main problem in this type of platform is task distribution among different processors. They proposed an algorithm [12] for resource allocation to improve the detection and avoidance of deadlock in heterogeneous platforms. The authors experimented with Resource Allocation Graph (RAG) and CloudSim, reducing the matrix complexity of the RAG for better efficiency in distributed systems like the cloud.

The article "Deadlock Avoidance in Sequential Resource Allocation Systems with Potential Resource Outages" by Spyros Reveliotis and his colleagues [13], published in 2016, aimed to extend the Sequential Resource Allocation Systems (RASs) by addressing potential resource outages. They used the Switched Discrete Event Systems (s-DES) to systematically treat the complicated version of the RAS deadlock avoidance problem. They decomposed the RAS operation and corresponding policy into operational modes

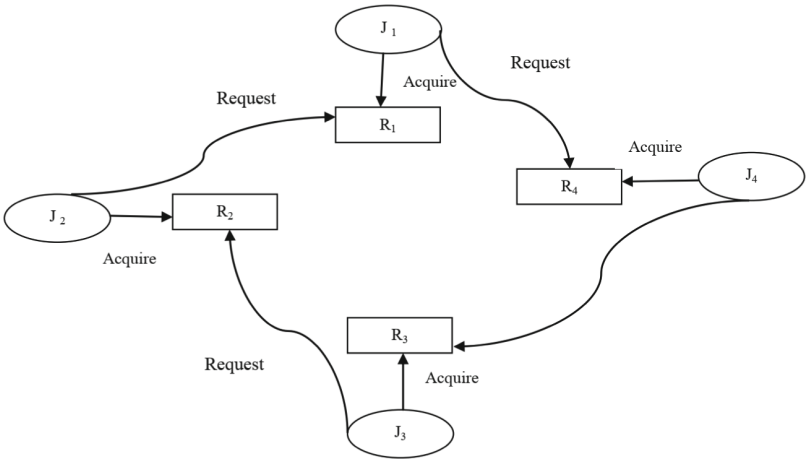

Fig. 4. Deadlock in Cloud [10].

and developed localized predicates to enable the formal characterization and effective computation of the sought supervisor, which leads to a distributed representation for the supervisor suitable for real-time implementation.

Deepti Malhotra [14] developed an efficient control scheme in 2016 to prevent deadlock in grid computing, which is difficult to avoid or detect. The objective of the study was to maximize resource availability and utilization to prevent deadlock and preserve data consistency. Using the C programming language, the proposed algorithm was tested and shown to detect deadlock and unstable states in virtual machines. In 2018, Emeka E. Ugwuanyi [15] proposed a resource provisioning algorithm using banker's deadlock avoidance technique for Industrial IoT devices in Multi-Access Edge Computing (MEC), which reduces communication overhead by incorporating SDN. The simulation results of the paper showed that applying the proposed algorithm prevents system deadlock and leads to a more reliable network interaction between mobile stations and MEC platforms (Table 1).

Table 1. Deadlock strategies [15]

Detection Algorithms	Lamport's algorithm
	Chandy-Misra-Haas algorithm
	Parallel Deadlock Detection Algorithm
	Detection in heterogeneous systems
	Unstructured deadlock detection
Prevention Algorithms	Load balancing methods
	Deadlock Prevention Algorithm in Grid Environment
Avoidance Algorithms	Banker's algorithm

In 2019, Cuong Nguyen and colleagues [16] introduced a novel model for resource allocation, called Avoid Deadlock Resource Allocation (ADRA). The authors aimed to improve resource allocation strategies for cloud computing, which inherited the concept of grid computing. The ADRA algorithm uses Wait-For-Graph (WFG) to prevent deadlock by allocating multiple resources to competing services running on heterogeneous distributed platforms. The algorithm was evaluated in a simulation environment using CloudSim, and its complexity was analyzed, revealing an improvement in time complexity compared to previous approaches. The experimental results showed that the ADRA algorithm is efficient and effective in managing resources and preventing deadlock.

In 2019, Sonam Sherpa et al. [17] presented a new approach to understanding and detecting deadlocks in software development. They focused on the perspective of developers and programmers who create the algorithms that can lead to deadlock. The authors argued that it is difficult to diagnose and reproduce deadlock bugs during the algorithm development process, resulting in high runtime overhead. To address this problem, they proposed the use of resource consumption footprints to identify concurrency bugs. This approach is based on the idea that the patterns of resource access and consumption are critical indicators of the behavior of concurrent software and can be used to guide the software debugging process. The authors demonstrated that monitoring resource footprints at runtime can effectively help detect software bugs, and for MPI programs, a simple SVM classifier can detect deadlocks with high accuracy using only the CPU usage patterns. The paper provides a different perspective on the problem of deadlock and suggests the use of machine learning techniques to detect and avoid it. Figure 5 is the testing result of Sonam Sherpa et al. which shows that the CPU usage can be reached to 100%.

In 2021, Yu V Bondarenko and colleagues [18] published a study on an algorithm and model to enhance the avoidance of deadlock in cloud environments. The objective was to improve the efficiency of resource allocation for the dynamic scaling of services and virtualized resources. As resource constraints can lead to deadlock, the authors designed an algorithm that leverages the execution time attribute of the process and a checking system to ensure a safe state and avoid deadlock. The improved algorithm builds on the banker algorithm by incorporating four arrays: MaxV for maximum available resources in VM, AlocV for allocation process in the initial state, AvelV for available resources in the initial state, and TQ for a temporary Queue to save the ID of waiting job requests. The paper provides a mathematical model and shows results obtained from the modeling.

After reviewing past research on deadlock in cloud environments, we have identified a gap in the field and propose a new prediction method to address it. Our method uses data on the virtual machines (VMs) and their resources to predict the likelihood of a deadlock occurring. This prediction is then integrated into the load balancing process, allowing requests to be allocated to the most appropriate VM for optimal performance. Importantly, our approach is fully automated and does not require input from expert analysts.

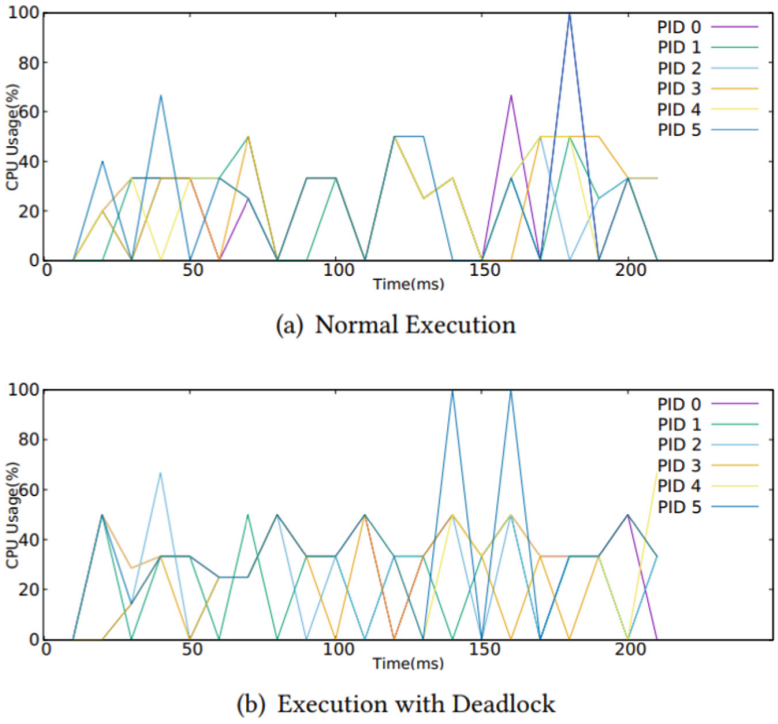

Fig. 5. Resource Consumption Traces [17]

3 Proposal

Our proposal focuses on ensuring good responsiveness and avoiding deadlock in the cloud environment through the use of algorithms for deadlock prediction and probabilities. We aim to improve service for users by ensuring a safe state. Our approach specifically targets virtualization in the cloud, utilizing virtual machines (VMs) and VM pools on physical hosts and datacenters. We believe it is necessary to focus on potential deadlocks in VMs, as they are responsible for processing and handling all user requests.

3.1 Research Model

The resource allocation algorithm is not suitable for systems that have multiple instances for each resource type. However, the proposed algorithm for avoiding deadlock can be implemented in load balancing for cloud environments. The algorithm aims to predict and detect deadlock occurrences by identifying busy and active areas and reallocating requests to more available resources. The algorithm is called DPA, which stands for Deadlock Prediction Algorithm. The algorithm utilizes Linear Regression prediction to forecast the unsafe state of VMs and the possibility of deadlock. Based on this prediction, the algorithm allocates requests to available resources to prevent deadlock from occurring. Figure 6 illustrates our research model which is less simple than the real cloud environment model.

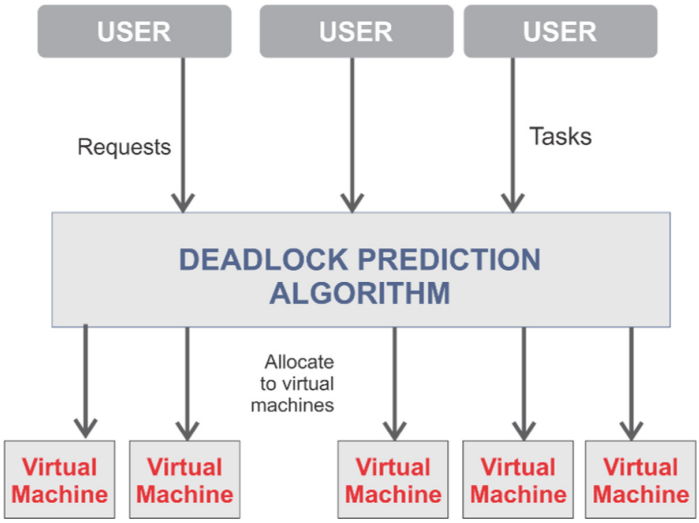

Fig. 6. The proposed DPA algorithm is developed and tested within the context of a research model for the cloud environment.

The Deadlock Prediction Algorithm (DPA) is developed based on the characteristics of deadlock in the Cloud research model. Deadlock is caused by tasks or requests that have processing times that are too long, leading to timeouts and disconnections. It can also occur when cloud resources are in conflict, where numerous requests or tasks are accessing the same resource at the same time, creating an endless loop that results in system crashes. The most common resources that are vulnerable to deadlock are the MEMORY (RAM), CPU, and STORAGE.

3.2 Proposed Algorithm

When a new request, or "cloudlet," is added to the Cloud, it comes with parameters such as size, number of files, return type, and more, which require a corresponding amount of resources to handle it. These resources are the virtual machines' resources and must not exceed the total available resources in the Cloud system. The system must first determine if allocating these resources will leave it in a safe state before proceeding with allocation. If it's safe, the resources are allocated, but if not, the process will have to wait until sufficient resources are released or be assigned to another virtual machine's resources.

To maintain a safe state and prevent deadlock, a prediction method is proposed based on calculating the probability of an unsafe state. This is achieved by using linear regression on historical data of past requests to build a prediction model. The input data includes request size, output size, file size, and the number of processors required, while the output data includes the usage of CPU, MEMORY, and STORAGE as percentages. If the predicted output values meet certain conditions, such as being less than the allowable thresholds, the request will be allocated to a virtual machine with suitable resource usage.

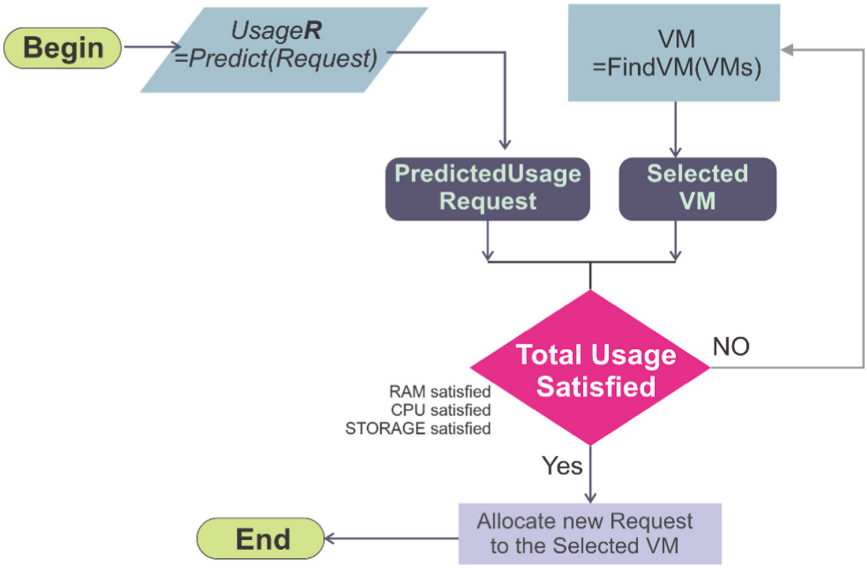

Fig. 7. Operating Diagram of the proposed algorithm DPA.

Figure 7 illustrates the proposed DPA, which comprises three main functions: 1) resource usage prediction function, which utilizes historical dataset and Linear Regression to predict resource usage; 2) select available VM function, which selects the VM with the minimum resource usage status among available VMs; and 3) allocation function, which allocates the request to the selected VM.

Pseudocode of DPA

```
1.  For each Request in CloudRequests
2.    isLocated = false;
3.    PredictedUsage = {RAM, CPU, STORAGE}predicted = Predict(Request);
            ← Function 1: regression on historical dataset
4.    VM = FindVM(VMList);
            ← Function 2: select the VM with least resources usage
5.    If isSatisfied(PredictedUsage, VM)
6.          AllocateRequestToVM(VM, Request);
            ← Function 3: allocate the Request to the VM
7.          isLocated = true;
8.    End If
9.    If(!isLocated)
10.     VM = VMList.getSelectedVM();
11.     AllocateRequestToVM(VM, Request);
12.   End If
13. End For
```

This pseudocode outlines the steps of the proposed algorithm (DPA) for allocating resources to cloud requests in order to avoid deadlock. For each request in the list of CloudRequests, the algorithm first predicts the required resources (RAM, CPU, and

STORAGE) using historical data and a regression model. It then selects a virtual machine (VM) with the least resource usage that can satisfy the predicted resource requirements. If such a VM is found, the request is allocated to it. If not, the algorithm selects a VM based on a pre-defined selection criteria and allocates the request to it. The algorithm continues this process until all requests have been allocated to VMs. The proposed algorithm can be described in the following steps:

Step 1: The Deadlock Prediction Algorithm (DPA) maintains and updates information about the resource usage status of all virtual machines (VM) in real-time for load balancing. This information is stored in the Load Balancer (LB), which tracks the usage levels of each VM for MEMORY (RAM), CPU, and STORAGE, calculated as a percentage. Initially, all VMs are listed in the "VMUsageList" table with usage figures set to 0.

$$Usage = \{RAM,\ CPU,\ STORAGE\} \tag{1}$$

Step 2: When the load balancer receives a new request, it needs to process the task using appropriate resources. To predict which resources to use for the current request, the algorithm employs a Linear Regression Model that is built on the data of previous requests. If the algorithm is being initialized, the original dataset selected from the natural load balancing in Cloud is used for this purpose.

$$Predicted\ Usage = \{RAM,\ CPU,\ STORAGE\}_{predicted} = Predict(Request) \tag{2}$$

Step 3: The DPA load balancer is queried by the Central Command Control (CCC) for the next allocation. The corresponding resource usage for the current request is determined to allocate it to the virtual machine (VM) only if three conditions are met simultaneously. The RAM threshold is set between 90% and 95% of total RAM usage in the VM, depending on its power and configuration. The CPU threshold is set at a slightly higher level than the RAM threshold, within the range of 97% to 98%. The STORAGE threshold is set between 96% to 99%. If any of these three conditions is not met, the LB will reject the VM and try to find another suitable VM for allocation.

$$PredRAM\_UsageRequest + RAMUsageVM\ < ThresholdRAM \tag{3}$$

$$PredCPU\_UsageRequest + CPUUsageVM\ < ThresholdCPU \tag{4}$$

$$PredSTOR\_UsageRequest + STORUsageVM\ < ThresholdSTOR \tag{5}$$

Step 4: DPA load balancer selects the VM with the smallest usage and sends its ID to the Central Command Controller (CCC) to process the request. The load balancer updates the VM ID and waits for new requests. If no VMs are initialized, the load balancer returns -1 to the CCC and queues the request for the next allocation.

Step 5: After processing the request on the virtual machine (VM), the response is sent to the central controller (CCC), which will then notify the DPA load balancer. The load balancer will update the "*vmUsageList"* table and the Usage list of the VMs. If

there are multiple requests, the central controller will perform Step 3 repeatedly until all the requests are processed properly.

Evaluation Criteria
In the experiment, the cloud was simulated with the given parameters, and CloudSim's load balancing algorithms including Round Robin, MaxMin, MinMin, and FCFS were executed with the same inputs. The outputs were compared, especially the Response Time parameters such as average, maximum, and minimum, as well as the total processing time (Makespan). The effectiveness of the evaluated algorithm is better if the predicted response time of the virtual machines and the predictive response time of the cloud have less error, which can lead to lower costs. The accuracy of the Linear Regression Model was evaluated using the RAE (Relative Absolute Error).

4 Simulation

This paper utilizes the CloudSim [26] library, written in the JAVA programming language, to simulate a Cloud environment. Additionally, the Weka library [27] is integrated into the simulation environment using JAVA, which allows for the use of the built-in LinearRegression function.

4.1 Simulation Environment

The Cloud environment used for the experiment consists of one Datacenter that has five hosts running five virtual machines. To test the effectiveness of the proposed Predicting Deadlock Occurrence Algorithm (DPA), random requests with different resource requirements are created (Tables 2 and 3).

Table 2. Configuration of Cloud Environment.

Datacenter configuration	Host and VM
- Number of hosts in datacenter: 5 - arch: x86 - OS: Linux - VMM: Xen - Time Zone: + 7 GMT - Cost: 3.0 - Cost per Memory: 0.05 - Cost per Storage: 0.1 - Cost per Bandwidth: 0.1	Each host in Datacenter has the following configuration: - CPU with 4 cores, each core is 1000 mips - Ram: 16384 (MB) - Storage: 1000000 Megabytes - Bandwidth: 10000 Mbps Each VM has the following configuration: - Size: 10000 MB - Ram: 512MB - Mips: 250 - Bandwith: 1000 Mbps - Pes no. 1 - VMM: Xen

Table 3. Request variant.

Length (MB)	File Size (Byte)	Output Size (Byte)	PEs
3000–1700	5000–45000	450–750	1

The proposed algorithm has been implemented in CloudSim's open source code by creating a class called *DPASchedulingAlgorithm* that inherits from the *BaseSchedulingAlgorithm* object. Several methods and properties have been updated to calculate virtual machine usage, and built-in functions have been adjusted to align with the proposed algorithm. The *predictRequestUsage* function has been added to forecast the resource usage of the request. Moreover, the *getMostSuitableVM* function is utilized to select a virtual machine that satisfies the three threshold conditions for request allocation.

4.2 Simulation Results

We performed simulation experiments using three different sets of input data, with 24, 100, and 997 requests. The request data was generated using *Epigenomics*, a tool proposed by https://github.com/WorkflowSim/WorkflowSim-1.0 [28].

Please note that the first paragraph of a section or subsection is not indented. The first paragraphs that follows a table, figure, equation etc. does not have an indent, either.

The first experiment, called ***Case 1*** (Epigenomics_24), involved simulating 24 requests in the Cloud environment using the proposed DPA algorithm and other load balancing algorithms. Table 4 and Fig. 8 display the results. Although the DPA algorithm seems to perform slightly better in terms of having the lowest average processing time, the difference between the algorithms is not significant due to the small number of requests. The FCFS algorithm, which processes requests in the order they are received, performed the worst with consistently higher processing times.

Table 4. Comparing the response times of algorithms in case 1.

Case 1	*Overall response time*		
	Avg (ms)	*Min (ms)*	*Max (ms)*
FCCS	1993.833	***0.18***	20363.49
MaxMin	1632.78	0.11	12825.62
Round Robin	1474.61	0.10	9487.84
MinMin	930.43	0.10	6027.47
DPA	**915.34**	0.11	**4582.72**

For ***Case 2*** (Epigenomics_100), a total of 100 requests were tested, and the outcome is illustrated in Table 5 and Fig. 9. The experiment revealed that the DPA algorithm

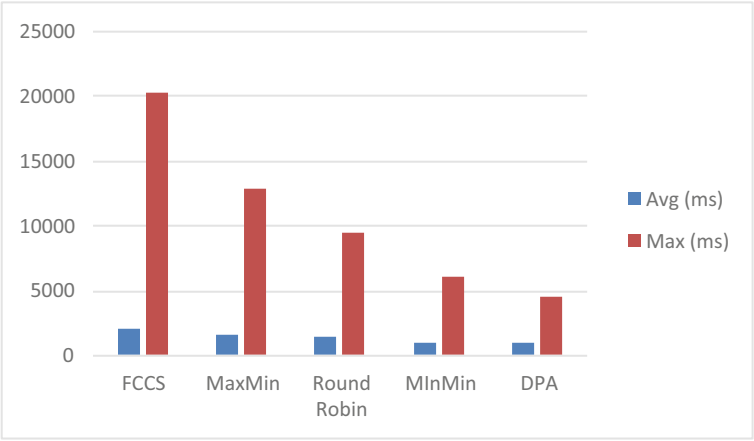

Fig. 8. Comparing the total response times of algorithms in case 1 with 24 requests.

outperforms other algorithms in terms of average processing time, while MaxMin has the lowest maximum processing time. Nonetheless, because of the limited number of requests, the variance between the algorithms is not significant. Additionally, the FCFS algorithm exhibits a natural pattern, with the processing time consistently the highest.

Table 5. Comparing the response times of algorithms in case 2.

Case 2	*Overall response time*		
	Avg (ms)	*Min (ms)*	*Max (ms)*
FCFS	6352.1466	0.11	139362.03
MaxMin	7,312.98	0.10	76,327.82
Round Robin	9,766.91	0.25	99,698.84
MinMin	10,158.29	0.11	407,527.17
DPA	6,659.82	0.12	76,659.82

In ***Case 3***, which involves 997 available requests in CloudSim, the results are presented in "Fig. 10" and "Table 6". The DPA algorithm performs best with the lowest average processing time and the lowest maximum processing time. As the number of requests increases, the predictive power and processing ability of the algorithm become more evident. The FCFS algorithm consistently shows the highest processing time.

The simulation experiments showed that the DPA algorithm performed slightly better than conventional algorithms in certain input data cases. In terms of response time and total processing time, it was not inferior to the existing algorithms and consistently performed better than them (Table 7).

From Fig. 11, we can see that in all three cases, the FCFS algorithm has the highest total execution time among all the algorithms. This means that FCFS takes the longest

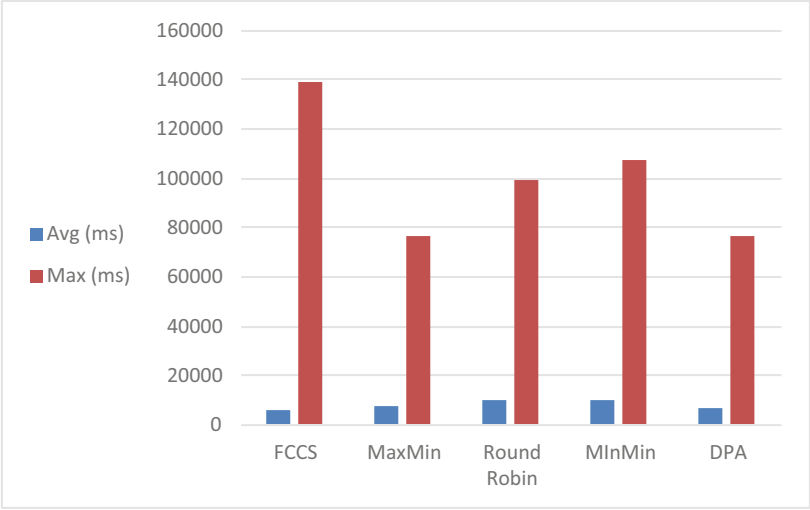

Fig. 9. Comparing the total response times of algorithms in case 2 with 100 requests.

Table 6. Comparing the response times of algorithms in case 3.

Case 3	*Overall response time*		
	Avg (ms)	*Min (ms)*	*Max (ms)*
FCFS	12927.32	0.13	380441.42
MaxMin	6,087.25	0.10	74,719.00
Round Robin	6,476.19	0.10	200,066.70
MinMin	6,455.54	0.11	204,196.72
DPA	5,940.80	0.11	57,559.75

time to process the requests in each case. In case 1, we can see that the MinMin algorithm has the lowest total execution time, followed by Round Robin, MaxMin, DPA, and FCFS, respectively. This suggests that in a scenario with a smaller number of requests, MinMin is the most efficient algorithm to use. In case 2, we can see that DPA has the lowest total execution time, followed by MinMin, Round Robin, MaxMin, and FCFS, respectively. This indicates that for a larger number of requests, DPA is the most efficient algorithm to use. In case 3, we can see that DPA again has the lowest total execution time, followed by MaxMin, MinMin, Round Robin, and FCFS, respectively. This suggests that as the number of requests increases further, DPA is still the most efficient algorithm to use. Overall, the chart indicates that the proposed DPA algorithm consistently outperforms

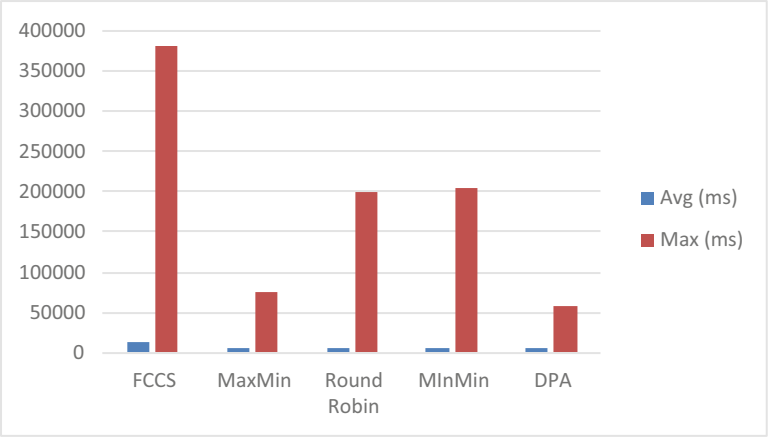

Fig. 10. Comparing the total response times of algorithms in case 3 with 997 requests.

Table 7. Comparing the Makespan of algorithms in all 3 cases

	Makespan (ms)		
	Epigenomics 24	*Epigenomics 100*	*Epigenomics 997*
FCFS	23,155	146,129	2,719,720
MaxMin	14,546	170,876	1,236,725
Round Robin	11,055	157,786	692,308
MinMin	7,593	414,381	1,303,421
DPA	6,297	152,507	1,206,652

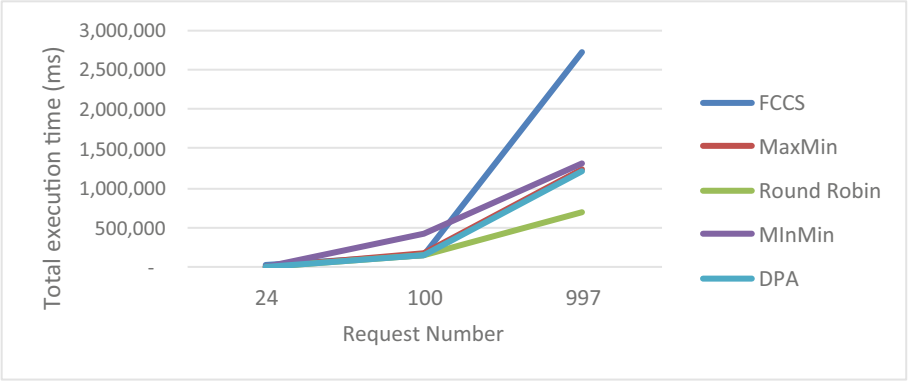

Fig. 11. Comparing the total execution time (makespan) of 5 algorithms in 3 cases.

the other algorithms in terms of total execution time, especially in scenarios with a larger number of requests.

Evaluation Linear Regression Model in DPA
In order to assess the performance of the Linear Regression Model used in the DPA algorithm, the Relative Absolute Error (RAE) metric was used to determine how accurately the model was able to predict the load balancer's values. The results are shown in Table 8, which indicates that the worst RAE value was obtained in predicting CPU usage in case 1, while the best value was obtained for Storage Usage prediction in the same case. Overall, the RAE values obtained in this experiment were deemed acceptable, although they varied depending on the request. However, the model's performance was not good in all cases due to the variability in requests.

Table 8. Comparing the RAE of DPA in 3 cases

	RAE metrics		
	Case 1 (24 requests)	Case 2 (100 requests)	Case 3 (997 requests)
CPU	0.565217	0.489868	0.324261
MEMORY (RAM)	0.148717	0.286780	0.310741
STORAGE	0.002531	0.006759	0.495361

From the RAE table, we can interpret that the prediction model performs better for CPU resource usage compared to Memory and Storage resources in all three cases. The RAE value for CPU decreases as the number of requests increases, indicating that the accuracy of the prediction model increases with more data. However, the RAE value for Memory increases in case 2, indicating a decrease in accuracy, while the RAE value for Storage is significantly higher in case 3, indicating poor accuracy in predicting Storage resource usage. Therefore, the prediction model needs improvement for predicting Memory and Storage resource usage accurately, especially in large-scale Cloud environments with a higher number of requests.

5 Conclusion

The main focus of this paper is to investigate deadlock and its avoidance in cloud computing using prediction algorithms, to ensure the safety of the cloud environment. The goal is to prevent deadlock occurrence so that the load balancer can allocate tasks to the appropriate virtual machine without encountering any deadlock issues. Through analyzing existing algorithms and previous studies, the researchers were able to gain a better understanding of the deadlock problem and identify the pros and cons of each algorithm. This led to the proposal of a new algorithm, the DPA algorithm, which improves load balancing in the cloud by improving response time, using limited resources, and providing more powerful virtual machines to handle requests more efficiently. The simulation

experiments showed that the DPA algorithm outperformed the Round Robin, MaxMin, MinMin, and FCFS algorithms. This paper achieved the objectives of reviewing the key issues and problems of deadlock and deadlock avoidance in cloud environments, and proposing a promising approach to improve load balancing efficiency in practical cloud environments.

The paper has some limitations which include that the proposed algorithm has not been applied in a physical cloud yet. Additionally, the improvements in response time and processing time are minor. To address these limitations, future research could explore the use of additional prediction techniques and optimization methods for the proposed algorithm. It would also be beneficial to apply the proposed algorithm in practical applications instead of just in a simulation environment.

Acknowledgment. We would like to express our big appreciation to institutions: Posts and Telecommunication Institute of Technology Ho Chi Minh City, Vietnam and Ho Chi Minh City Open University, Ho Chi Minh City, Vietnam.

References

1. Wen, Y.-F., Chang, C.-L.: Load balancing job assignment for cluster-based cloud computing. In: 2014 Sixth International Conference on Ubiquitous and Future Networks -ICUFN (2014)
2. Shao, G., Chen, J.: A load balancing strategy based on data correlation in cloud computing. In: Proceedings of the 9th International Conference on Utility and Cloud Computing (2016)
3. Mishra, S.K., Sahoo, B., Parida, P.P.: Load balancing in cloud computing: a big picture. J. King Saud University – Computer and Information Sci. (2018)
4. Shahid, M.A., Islam, N., Alam, M.M., Su'ud, M.M., Musa, S.: A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access **8**, 130500–130526 (2020)
5. Iqbal, S., Kiah, M.L.M., Anuar, N.B., Daghighi, B., Wahab, A.W.A., Khan, S.: Service delivery models of cloud computing: security issues and open challenges: cloud computing security. Security and Communication Networks **9**(17), 4726–4750 (2016)
6. Tekale, S., Britto, J.G.M., Gousia Banu, A.S.: Load balancing in cloud computing. International J. Engineering and Advanced Technology **8**(6S3), 2164–2166 (2019)
7. Shah, N., Farik, M.: Static load balancing algorithms in cloud computing: challenges & solutions. International Journal of Scientific & Technol. Res. **4**(10), 365–367 (2015)
8. Rashmi, K.S., Suma, V., Vaidehi, M.: Enhanced Load Balancing Approach to Avoid Deadlocks in Cloud (2012)
9. Lim, J., Suh, T., Yu, H.: A deadlock detection algorithm using gossip in cloud computing environments. In: Lecture Notes in Electrical Engineering, Springer Netherlands, Dordrecht, pp. 781–789 (2013)
10. Mahitha, O., Suma, V.: Deadlock avoidance through efficient load balancing to control disaster in cloud environment. In: 2013 Fourth International Conference on Computing, Communications and Networking Technologies – ICCCNT (2013)
11. Ha Huy Cuong Nguyen, V.S.L.: Detection and avoidance deadlock for resource allocation in heterogeneous distributed platforms. International Journal of Computer Science and Telecommunications **6**(2) (2015)
12. Nguyen, H.H.C., Dang, H.V., Pham, N.M.N., Le, V.S., Nguyen, T.T.: Deadlock detection for resource allocation in heterogeneous distributed platforms. In: Advances in Intelligent Systems and Computing, Springer International Publishing, Cham, pp. 285–295 (2015)

13. Reveliotis, S., Fei, Z.: Robust deadlock avoidance for sequential resource allocation systems with resource outages. In: 2016 IEEE International Conference on Automation Science and Engineering - CASE (2016)
14. Malhotra, D.: Deadlock prevention algorithm in grid environment. MATEC Web Conference **57**, 02-013 (2016)
15. Ugwuanyi, E.E., Ghosh, S., Iqbal, M., Dagiuklas, T.: Reliable resource provisioning using bankers' deadlock avoidance algorithm in MEC for industrial IoT. IEEE Access **6**, 43327–43335 (2018)
16. Nguyen, H.H.C., Doan, V.T.: Avoid deadlock resource allocation (ADRA) model V VM-out-of-N PM. International Journal of Innovative Technology and Interdisciplinary Sciences **2**(1), 98–107 (2019)
17. Sherpa, S., Vicenciodelmoral, A., Zhao, X..: Deadlock detection for concurrent programs using resource footprints. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion - UCC'19 Companion (2019)
18. Bondarenko, Y.V., Azeez, A.E.: Algorithm and model for improve the avoiding of deadlock with increasing efficiency of resource allocation in cloud environment. J. Physics - Conference Series **1902**(1), 012–054 (2021)
19. Almhanna, M.S., Almuttairi, R.M.: Chapter 6 methods for handling deadlocks. In: Operation System, University of Babylon (2019)
20. da Silva, R.F., Juve, G., Rynge, M., Deelman, E., Livny, M.: Online task resource consumption prediction for scientific workflows. Parallel Processing Letters **25**(03), 15–41 (2015)
21. Matsunaga, A., Fortes, J.A.B.: On the use of machine learning to predict the time and resources consumed by applications. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (2010)
22. Witten, I.H., Frank, E., Hall, M.A.: Data Mining.: Practical Machine Learning Tools and Techniques, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2011)
23. Salzberg, S.L.: C4.5: Programs for machine learning, J. ross Quinlan. Morgan Kaufmann publishers, inc., 1993, Machine learning **16**(3), 235–240 (1994)
24. Monge, D.A., Holec, M., Železný, F., Garino, C.G.: Ensemble learning of runtime prediction models for gene-expression analysis workflows. Cluster Computing **18**(4), 1317–1329 (2015)
25. Walczak, S., Cerpa, N.: Artificial neural networks. In: Encyclopedia of Physical Science and Technology, Elsevier, pp. 631–645 (2003)
26. Abdulkareem, D., Noor, Z.J., Abdullah, A.: CloudSim 3.0.3 Simulator Step by Step. Unpublished (2021)
27. Weka 3 - data mining with open source machine learning software in java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed 27 Apr 2022
28. WorkflowSim-1.0: Wiki pages. https://github.com/WorkflowSim/WorkflowSim-1.0. Accessed 27 Apr 2022