Algorithm and model for improve the avoiding of deadlock with increasing efficiency of resource allocation in cloud environment

Yu V Bondarenko, A E Azeez
Department of Applied Mathematics, Informatics and Mechanics, Voronezh State University, 1, University Square, Voronezh, 394018, Russia
E-mail: bond.julia@mail.ru, ammar.azeen548@gmail.com

Abstract. Recently, cloud computing has been the most popular method among all other promising technologies that provides the service for many users at the same time. all users can reach and use the resource that is provided by the Virtual Machines (VMs). Hence users process, data must arrive to the resources to perform as a disturbed system, which provides dynamical scale services and virtualized resources. As a result of limitation of resources, the deadlock status may occur in such type of systems. In this paper, we have designed an algorithm that could increase the efficiency of deadlock avoidance. Our technique will use the attribute of execution time in the process to get better resource allocation in additional to the checking system to stay in safe state and free of deadlock.

1. Introduction
In recent years we have seen a high increase in the data that need to be processed this lead to improve the new technologies using distributed computing to reduce the cost to process such a high amount of data as well as a lot of researches appears published in this direction, so as a result for this turnout pushed markets for high investment in this field, which led to its rapid development [1], millions of users now just depends on this technology such as cluster computing and Cloud computing. In this research, will focus on the Cloud computing which provide multi types of resources and different services, it could be by providing hardware such as CPU performance, memory capacity, etc. this type of service is called infrastructure as a service (IaaS), which is so popular in use (e.g., Amazon EC2) [2]. And there are other types of cloud services which provides a capacity of Cloud resources, platform as a service (PaaS) or (SaaS) that refer to software as a service [3]. Furthermore, the resources need to be managed by the management of resource allocation available in Cloud and dynamically VMs resource allocation using strategies match with Quality of Service (QoS). keeping the VMs physical resource work in high performance [4]. Hence, the virtualization technology made VMs ability to use cluster system and work like separated physical hardware although its sharing [5], while here we have shared resources so there is a possibility of deadlock there is a lot of algorithms that had designed just for avoiding this problem. But all these algorithms do the job form one side is to keep system far from deadlock state [6]. In such status we suggest in our work to use the time of process to keep the Cloud faster response to share resource and allocate the resource efficacy and the system always in a safe state.
2. Algorithms for avoiding the deadlock
In the first part we review the concepts of deadlock avoidance have already used. By surveying most of Previous studies. These studies concentrate on the deadlock avoidance policies (DAPs) [7], the system that control this operation is resource allocation system (RAS) [7]. Although, this strategies and algorithms developed a complex method to detect and deals with deadlock problem in all types of systems, most of these strategies are still depend on calculating the available resource framework and how many requests must be served only. For example, to avoid the deadlock the system may use Petri Net (PN). PN is one of the effective ways to deals with resources failure, in order to avoid deadlock [8]. The other popular method is Load Balancing, which used in different levels of cloud computing [9]. The primary problem solves by this paper is the deadlock avoidance in Cloud Computing with keeping higher level of Quality of Service (QoS). The (QoS) of the cloud under all these actions may cause a lot of delay of execution time for jobs., we improved and proposed an algorithm using the technique of detect and avoid the deadlock before it performed in the VMs. this algorithm will aim to eliminate the delay of time processing and reach save system.

3. Deadlock model design
The distributed system can be modeled by mathematical logic. However, the detection of deadlock directly from equations is not clear, by minimize the function of probability of deadlock and represent the VMs as a matrix VM_{ij} [9], if we conceder we have n of VMs and the total capacity of each denoted by C_{ij} so:

\[ \sum_{i}^{n} C_{ij} = \sum_{i}^{n} A_{ij} + \sum_{i}^{n} R_{ij}. \]

Where A is available resources, and R is Already Allocated resource, so the new job request can be performed only if the need matrix of request job less than or equal to available matrix of resources as following:

\[ R_{ij} \geq A_{ij}. \]

While the R_{ij} is a mount of resource that process need. Therefore, the graphical methods to detect and solve the problem of deadlock is the most common and efficient. In order to achieve the results of existing algorithms and proposed algorithms, we have a model example for the deadlock problem. Although our work will deal with resource allocation in cloud computing, this technology uses the concept of a smart distributed system that provides different types of resource allocation and sharing VMs. The cloud computing and the traditional distributed architecture share a similar function that is the pool of multi-access to resources by requesting numbers of jobs to process at the same time. In this case, the quality of cloud will depend on providing the best way of resource allocation and number of processing that is submitted into the cloud [10]. In a heterogeneous-distributed systems, there is nonsynchronous processors \( P_1, P_2, P_3, \ldots, P_k, \ k \leq n \). All these processes have a different capacity and different global memory because the system becomes as a resource of a shared memory. In this system, the important part is the communication network between the resources itself and the jobs that requested these resources. This allows the jobs to be able to share the physical resources (CPU cores, amount of memory, etc.) and leads to accomplish the jobs in a minimum cost [5]. When the job has a reservation for any resource in VM, the other jobs must wait until this resource becomes free. As a result of this process, this may lead to a deadlock just like in an operating system. To demonstrate the proposed algorithm, the status and deadlock problem will be presented by a simple regular graph such as the Resource Allocation Graph (RAG) and Wait For Graph (WFG) [11]. The WFG is a standard tool for deadlock detection. The proposed model considers that there is a number of jobs that required a number of resources from VM.
Every job will have an ID that relates to the VM ID that requires the resource. In this case, the algorithm must detect all the attributes of the job and the VM status.

![Virtual machines pool](image1)

**Figure 1.** The main structure for resource allocation in cloud computing.

Figure 1 illustrates an example of multi requesting jobs to service by three VMs. Each VM has different capacity of CPU and each job has variant time of execution. Figure 1 also shows that a required tow reoccurs may occur for the same job. Simply, the deadlock may happen where the job requests for resource that already held by another job that still running. This process method will make both of processes in waiting status, and it will lead to the fact that the deadlock in cloud has the same four conditions of deadlock in operating system [11]:

1. Mutual Exclusion: when the resource is not sharable nor have enough space or energy to process more than one job at the same time.
2. Hold and wait: means the process (job) is holding on resource and at the same time waiting for another resource, but the resource that is already waiting for is held by other process.
3. No Preemption: the process can only release the resource because the resource can’t do this by itself.
4. Circular waiting: It is the most common condition when each process waits for held resources.

The Resource Allocation Graph is representing the deadlock problem, and it’s not deferent from (WFG) which is more focusing on the problem. Let us assume that we have a set of processes each process defined as \( P \) so a set will represent as:

\[
P = \{P_1, P_2, P_3, \ldots, P_k\}, \quad \text{while } k > 1,
\]

Simultaneously, the same \( P_i \) held resource \( (R_1) \) while it is waiting resource \( (R_2) \) to be released from the other process for example \( (P_{i+1}) \). At the same time, process \( (P_{i+1}) \) is waiting for \( P_i \) to release \( R_1 \). This status will generate a non-finite circle of waiting and this situation is represented by (WFG). In the Graph, we can see if the circle was broken by making some processes permanently waiting for period of time to get permission before the request, for example in figure 3 if we delayed the request of \( P_1 \) until \( P_2 \) release the \( R_1 \), and the same action with the requesting of \( P_3 \) that request resources of \( R_3 \) this will give time to \( P_2 \) to release the \( R_3 \), in this way we will never enter in the circle of deadlock. While the system is trying to avoid deadlock this effect on the QoS for the cloud. This problem will be the main objective for proposed algorithm.

To deal with the problem of deadlock in the virtualized computing, there are more than one technique including ignoring the deadlock, passing it, or resolving it after occur [12].
4. Proposed algorithm

In this algorithm, we proposed an approach that can be used to improve the exacting algorithms by reallocating resource in an efficient way by taking the consideration of time of execution for every job request. First, we assume that a set of resources \( R = \{ R_1, R_2, \ldots , R_m \} \), where \( m \) are all available resources in the system, and the capacity of each resource represented by \( C_R \) that refers to a number of instances of each resource \( R \) which is creating the matrix of \( C_{ij} \) for a total capacity of the system. In other words, how many tasks can the resource hold at a time depends on the cloud capacity [13]. The proposed algorithm suggests that performing of a requested task without ignore the time of execution \( (t) \) that every process \( (R) \) needs to release the resource \( (R) \). Each of these process in job Queue must declare how much its time for execution \( (t) \) before submission to the system, as a result of this statement our algorithm will use the concept of Short Job First (SJF) [11], but not the algorithm itself because in avoiding deadlock must use Algorithms for resource allocation to make the system in save status. So if there are two process requesting \( (P) \) from the job queue to use the resource \( R_1 \) to \( R_m \in R \), and our algorithm will check if \( P \leq C_R \) and compare the time of execution of both to be sure that will not cause a deadlock in next job submission otherwise the job will be set to waiting statues in temporary queue and the system submit other request and do the same audit in this way of choosing submission jobs will guarantee the QoS and all requesting process will be able to use the resources. as a result of our technic the total execution time for all requests will be faster and free of deadlock, this technic can be achieved with any type of resource allocation algorithm that Cloud use it. As showed in the figure 4. In this paper we will
achieve it as numerical example in Banker algorithm [14] and the Load Balancing methods, by adding the attribute of the processes itself as an important argument. This attribute will play a significant role to determine the priority of resource allocation and jobs execution in a cloud environment.

Figure 4. The flowchart of the proposed Algorithm.

4.1. Improve banker algorithm
As a practical example of the proposed algorithm. Applying banker algorithm, which is most faster was to a void deadlock using the strategy of banks, by keeping reserve resources [11].

4.1.1. Existing algorithm
1. Input: \( n \leftarrow \) number of process
2. \( m \leftarrow \) number of resources in VM
3. \( \text{MaxV}[] \leftarrow \) the maximum available resources in VM
4. \( \text{AlocV}[] \leftarrow \) the allocation process in initial state
5. \( \text{AvelV}[] \leftarrow \) the available resources in initial state
6. Begin
   6. Step 1: set index\([i]\) = 0 for \( \forall P \) where \( i = 0 \) to \( n - 1 \)
   7. Step 2: \( \text{NeedV}[n][m] = \text{MaxV}[n][m] - \text{AlocV}[n][m] \)
   8. Step 3: For \( \forall P \) find index\([i]\) = 0 and \( \text{NeedVi} \leq \text{AvelV} \)
   9. If true then index\([i]\) = 1
   10. Free the VM resource & \( \text{AvelV} = \text{AvelV} + \text{AlocV} \)
   11. Until \( \forall P \) find index\([i]\) = 1
   12. End for
   13. End if
   14. Step4: the system in safe state
   End

4.1.2. Improve by Proposed algorithm
1. Input: \( n \leftarrow \) number of process
2. \( m \leftarrow \) number of resources in VM
3. \( \text{MaxV}[] \leftarrow \) the maximum available resources in VM
4. \( \text{AlocV}[] \leftarrow \) the allocation process in initial state
5. \( \text{AvelV}[] \leftarrow \) the available resources in initial state
6. \( \text{TQ}[] \leftarrow \) temporary Queue to save the ID of waiting job request
7. \( t \leftarrow \text{total execution time for all process} \)
   Begin
8. Step 1: set index\([i]\) = 0 for \( \forall P \) where \( i = 0 \) to \( n - 1 \)
9. Step 2: NeedV\([n]\)[\(m\)] = MaxV\([n]\)[\(m\)] - AlocV\([n]\)[\(m\)]
10. Step 3: if there is more than resource request than start with the minimum \( t(P_i) \)
    So for \( \forall P \) arrange descending according to \( t(P_i) \)
12. Step 4: For \( \forall P \) find index\([i]\) = 0 and NeedV\(_i\) <= AvellV
13. Free the VM resource & AvellV = AvellV + AlocV
14. Else set \( P_i \) to TQ [ ]
15. Set \( t(P_i) \) to TQ [ ]
16. Until \( \forall P \) find index\([i]\) = 1
17. End for
18. End if
20. Step 5: if TQ [ ] \( \neq \) empty
21. Set all requests \( P \) only from TQ [ ]
22. Select the minimum \( t(P_i) \)
23. Go to step 4
24. Step 6: the system in safe state
   End

4.2. Implement proposed algorithm in load balancing method
As a second example for practically is by applying in load balancing, which is a very efficient
method to resource allocation in distributed systems. We get an efficient dynamic load balancing
scheduling algorithm.

4.2.1. Existing algorithm used load balancing
1. Input: \( n \leftarrow \) number of process
2. \( m \leftarrow \) number of resources in VM
3. set VMID \( \leftarrow \) to \( \forall \) VMs
   Begin
4. Step 1 : Receive request \( P_i \) by Data Control Centre
5. Step 2: for \( \forall \) VMID
6. If VMID=0 then
   Begin
7. Set \( P_i \) to execution in VM
8. Set VMID = 1
9. Else the Data Control Centre chick IF the \( P_i \) completed set VMID = 0 go to step 2
10. End IF
11. Step 3: load balancer receives new process and allocate to available VM (to VMID = 0)
12. Step 4: IF more than one VMID = 0 and more than \( P_i \) the load balancer will choose fist
    arrived to service by first VM available
13. Step 5: go to step 2 until the \( P_i \) =NULL

4.2.2. improve algorithm used load balancing
1. Input: \( n \leftarrow \) number of process
2. \( m \leftarrow \) number of resources in VM
3. set VMID \( \leftarrow \) to \( \forall \) VMs
4. set Temporary Queue \( \leftarrow \) TQ
   Begin
5. Step 1: Receive request \( P_i \) by Data Control Centre
6. Step 2: for ∀ VMID
7. IF VMID = 0 then
   Begin
8. Set \( P_i \) to execution in VM
9. Set VMID = 1
10. Else the Data Control Centre chick IF the Pi completed set VMID = 0 go to step 2
11. End IF
12. Step 3: load balancer receives new process and allocate to available VM (to VMID = 0)
13. Step 4: IF ∀ VMID = 1
   Begin
14. Set \( P_i \) in TQ with its execution time \((t)\) go to step 1
15. Step 5: IF more than \( P_i \) stored in the TQ
   Begin
16. Step 6: load balancer will choose the shortest process in execution time to service by first VM available
   End IF
   End IF
17. Step 7: go to step 2 until the \( P_i = NULL \)

So in simple explanation the proposed algorithm all requests will be served in the faster time as a result of the resource for example CPU will be available in maximum capacity faster when serve the short execution time request first.

5. Discussion of the result
We assume the numerical example of the algorithm as the following configurations:

|    | Execution Time(ms) | Allocation | Maximum | Available | Need |
|----|--------------------|------------|---------|-----------|------|
|    | \( P_0 \) | 140 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 5 | 2 | 0 | 0 | 0 | 0 | 0 |
|    | \( P_1 \) | 200 | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 | 1 | 5 | 3 | 2 | 0 | 7 | 5 | 0 |
|    | \( P_2 \) | 150 | 1 | 3 | 5 | 6 | 2 | 3 | 5 | 6 | 2 | 8 | 8 | 8 | 1 | 0 | 0 | 0 |
|    | \( P_3 \) | 400 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 | 2 | 14 | 11 | 10 | 0 | 0 | 8 | 0 |
|    | \( P_4 \) | 600 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 | 2 | 14 | 12 | 14 | 0 | 6 | 4 | 2 |

In table 1 we can observe that the requesting of the process for a resource can be continues as well as in the Cloud computing so in the begging we will make resource allocation by using the standard Banker algorithm and to avoid deadlock afar applying the algorithm we get the save sequence for this table as the flowing \( P_0, P_2, P_4, P_3, P_1 \) this sequence is a safe state for the system but the system ignored the time of execution for each process.

In proposed algorithm, we start with the short job first if don’t much the Banker so it will save in to Queue and then comparing the execution time for all process including that waiting in queue as a result by applying this technic in table 1 the process \( P_1 \) will be wait in the Queue then will get comparing between \( P_1, P_3 \) and \( P_4 \) so the save sequence will \( P_0, P_2, P_1, P_4, P_3 \). Here we notice that the waiting time for some process are reduce and total time in whole system was reduced this will be very useful for system of Cloud computing when we have unlimited for \( P_0 \)
Table 2. The waiting time for each process in system.

| Jobs request | Wait time for execution (ms) |
|--------------|-----------------------------|
| $P_0$        | 0                           |
| $P_2$        | 140                         |
| $P_4$        | 150                         |
| $P_3$        | 600                         |
| $P_1$        | 400                         |
| Total time   | 1290                        |

to $P_k$ while $k = \infty$ In this case the system will serve the shorter request as once it’s arrived after checking deadlock and in this way the total system efficiency will increase by free the resources as soon as possible to be able to get the more requests.

Table 3. The waiting time for each process in system after applying proposed algorithm.

| Jobs request | Wait time for execution (ms) |
|--------------|-----------------------------|
| $P_0$        | 0                           |
| $P_2$        | 140                         |
| $P_4$        | 150                         |
| $P_3$        | 200                         |
| $P_1$        | 600                         |
| Total time   | 1090                        |

Figure 5. Implementation efficiency of proposal algorithm.
6. Conclusion and future work
The response of the resource in the cloud computing is important because it increases the capability and the speed of task processing. Sometimes, the efficiency of the whole system is measured only by the processes speed. In the structure of the distributed system, the resource allocation way is one of the main processes, therefore; our proposed algorithm designs a new way to allocate the resource for the request process by finding the optimal way to choose the request that will serve first. Since this type of system dealing with multitasking processing, the problem of deadlock is likely to occur. Our work focuses on solving this problem with keep the improvement of QoS for Cloud by taking into consideration the timing. In future work, we suggest to add another process attribute like the importance of the request and use the parallel processing when the Cloud has enough resources for more than one process request.

References
[1] Ha Huy C and Van T 2015 International Journal of Computer Science and Information Security 13 (2) 1–6
[2] Amazon EC2 URL http://aws.amazon.com/ec2/
[3] Manvi S S and Shyam G K 2013 Journal of Network and Computing Applications 41 424–440
[4] Choudhary A, Shilpa Rana and Matahai K J 2016 Procedia Computer Science 78 132–138
[5] Warneke D and Kao O 2011 IEEE Trans. Distrib. Syst. 22 (6) 985–997
[6] Jiying Li Meikang Qiu, Jain-Wei Niu and YuChen Zhong Ming 2010 IEEE International Conference on Intelligent Systems Design and Applications 31–36
[7] Fei Z, Reveliotis S, Miremadi S and Akesson K 2015 IEEE Trans. on Automation Science and Engineering 12 990–1006
[8] Reveliotis S and Fei Z 2017 IEEE Trans Autom Sci Eng 14 (4) 1695–1711
[9] HC Nguyen H, Tran H D, Van T D and TP Anh V 2017 Lect Notes Inst Comput Sci Soc Telecommun Eng LNICST 172–193
[10] Nair Gopalakrishnan T R, M Vaidehi K S R and Suma V 2012 Proc. InConINDIA AISC 132 461–468
[11] Silberschatz A, Galvin P B and Gagne G 8th ed. Hoboken, NJ (USA: Wiley)
[12] Nguyen H and Le O 2015 Int. J. Comput. Sci. Telecommun. 6 (2) 40–49
[13] Wan C Wang C and Pei J 2012 Proceedings of the international conference on information science and technology China 634–639
[14] Ugwuanyi E E, Ghosh S, Iqbal M and DagiuKlas T 2018 IEEE Access 6 43327–43335