MINISTRY OF EDUCATION AND TRAINING

DA NANG UNIVERSITY

NGUYEN HA HUY CUONG

A RESEARCH ON DEADLOCK PREVENTION IN RESOURCE ALLOCATION FOR DISTRIBUTED VIRTUAL HOST SYSTEM

SPECIALIZATION: COMPUTER SCIENCE

CODE: 64.48.01.01

Supervisors

1. Associate Prof., Dr. Le Van Son
2. Prof., Dr. Nguyen Thanh Thuy

Da Nang - 2017

This thesis has been finished at:

Da Nang University of Technology

Da Nang University

Examiner 1: Associate Prof., Dr. Vo Viet Minh Nhat

Examiner 2:Associate Prof., Dr. Nguyen Thanh Binh

Examiner 3:Associate Prof., Dr. Ngo Hong Son

The PhD Thesis will be defended at the Thesis Assessment Committee at Da Nang University Level at Room No:

.....

.....

At date 18 month 01 2017

The thesis is available at: .....

1. The National Library.
2. The Information Resources Center, University of Da Nang.

LIST OF WORKS PUBLISHED

1. Nguyễn Hà Huy Cường (2012). Nghiên cứu giải pháp kỹ thuật ngăn chặn bế tắc trong cung cấp tài nguyên phân tán cho hệ thống máy chủ ảo, *Tạp chí Khoa học và Công nghệ, Viện Hàn Lâm Khoa học và Công nghệ Việt Nam*, **50**(3E), pp. 1324-1331.

2. Nguyễn Hà Huy Cường, Lê Văn Sơn, Nguyễn Thanh Thủy (2013). Ứng dụng thuật toán Kshemkalyani-Singhal phát hiện bế tắc trong cung cấp tài nguyên phân tán cho hệ thống máy chủ ảo, *Hội nghị Quốc gia lần thứ VI về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR)*, Huế, 20 – 21/6/2013, NXB Khoa học Tự nhiên và Công nghệ, Hà Nội, pp. 602-608.

3. Nguyễn Hà Huy Cường, Lê Văn Sơn (2013). Một chính sách hiệu quả cung cấp tài nguyên phân tán cho hệ thống máy chủ ảo, *Kỷ yếu Hội thảo quốc gia "Một số vấn đề chọn lọc của công nghệ thông tin và Truyền thông"*, Đà Nẵng, 14-15 tháng 11 năm 2013, NXB Khoa Học Tự Nhiên và Kỹ Thuật, Hà Nội, pp. 186-192.

4. Nguyễn Hà Huy Cường, Lê Văn Sơn (2014). Kỹ thuật cung cấp tài nguyên cho lớp hạ tầng IaaS, *Tạp chí Khoa học và Công nghệ*, Đại học Đà Nẵng, **7**(80), pp. 103-106.

5. Ha Huy Cuong Nguyen, Van Son Le, Thanh Thuy Nguyen (2014). Algorithmic approach to deadlock detection for resource allocation in heterogeneous platforms,*Proceedings of 2014 International Conference on Smart Computing*, 3-5 November, HongKong, China, IEEE Computer Society Press, pp. 97-103.

6. Ha Huy Cuong Nguyen, Dac Nhuong Le,Van Son Le, Thanh Thuy Nguyen (2015). A new technical solution for resources allocation in heterogenenous distributed plaforms, *Proceedings of 2015 The Sixth International Conference on the Applications of Digital Information and Web Technologies(ICADIWT2015)*, 10-12 Feb 2015, Macau, China, IOS Press, Volume 275, Issue 2, pp. 184-194.

7. Ha Huy Cuong Nguyen, Hung Vi Dang, Nguyen Minh Nhat Pham,Van Son Le, Thanh Thuy Nguyen (2015). Deadlock detection for resources allocation in heterogenenous distributed plaforms, *Proceedings of 2015 Advances in Intelligent Systems and Computing*, June 2015, Bangkok, Thailand, Spinger, Volume 361, Issue 2, pp. 285-295.

8. Ha Huy Cuong Nguyen (2016). Deadlock prevention for resource allocation in heterogeneous distributed platforms, *Proceedings of 2016 7th International Conference on Applications of Digital Information and Web Technologies*, 29-31 March 2016, Macau, China, IOS Press, Volume 282, pp. 40-49.

9. Ha Huy Cuong Nguyen, Van Son Le, Thanh Thuy Nguyen (2016). Deadlock Prevention for Resource Allocation in model nVM-out-of-1PM, *Proceedings of 2016 3th National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)* , 14-16 September 2016, The University of Da Nang, Viet Nam, IEEE Computer Society Press, pp. 247-252.

INTRODUCTION

In the past, grid computing and batch scheduling have both been commonly used for large scale computation. Cloud computing presents a different resource allocation paradigm than either grids or batch schedulers. In particular, Amazon C2 is equipped to, handle may smaller computer resource allocations, rather than a few, large request is as normally the case with grid computing. The introduction of heterogeneity allows clouds to be competitive with traditional distributed computing systems, which often consist of various types of architecture as well. Recently, reports have appeared that many of the studies provide cloud computing resources, the majority of this research to deal with variability in resource capacity for infrastructure and application performance in the cloud. We develop a method to prevent a deadlock occurs in the process of providing resources in class infrastructure as a service IaaS. Our rating indicates that the deadlock prevention method using two-way search algorithm may improve the effectiveness and efficiency of resource allocation for heterogeneous distributed platforms.

Resource allocation in cloud computing has attracted the attention of the research community in the last few years. The problem of request scheduling for multi-tiered web applications in virtualized heterogeneous systems in order to minimize energy consumption while meeting performance requirements. They proposed a heuristic for a multidimensional packing problem as an algorithm for workload consolidation. In previous articles we have published two algorithms. Which were used to detect deadlock in resources allocation heterogeneous distributed platforms. We provide deadlock detection algorithms and resolve the optimization problems of resources based the recovery of resources allocated. We provide deadlock detection algorithms and resolve optimal problems according to groups of users. Most of the studies were set to study scheduling policy effectiveness in resources allocation. Much of the research conducted homogeneous physical machines (PMs). Not much research provide resources for heterogeneous systems. To maximize performance, these scheduling algorithms tend to choose free load servers when allocating new VMs. On the other hand, the greedy algorithm can allocate a small lease (e.g. with one VM) to a multi-core physical machine. In this study, we propose solutions to detect deadlock in resource supply, then proceed to build the resource supply automatically detects and prevents deadlock occurs. This issue is also effective in allocation resources. The mathematical model computes the optimal number of servers and the frequencies. A new approach for dynamic autonomous resource management in computing clouds

introduced.

In this work, we propose a deadlock prevention algorithm, to prevent deadlock in providing resources for the virtualization heterogeneous distributed platforms. More specifically, our contributions are as follows:

1. We provide an algorithmic to prevent deadlock and resource allocation issues in the virtualization of heterogeneous distributed platforms. This algorithm is, in fact, more generally, even for heterogeneous distributed platforms, and only allows allocating minimal resources to meet QoS (*Quality of Service*) arbitrary force.

2. Based on the resource model provides P-out-of-Q. We develop the resource model provides multiple virtual machines on multiple physical machines scattered nVM-out-of-NPM.

3. We have studied the effects not effective in providing resources, predict failures may occur in the system and suggest different approaches to mitigate this problem, followed by set out a strategy of automation in providing resources.

The work is organized as follows: section 2 provides the related works; section 3 describes existing models; section 4 presents approaches and section 5 presents our conclusions and suggestions for future work finally.

Chương 1

OVERVIEW OF PREVENTION DEADLOCK IN RESOURCES ALLOCATION FOR SERVER DISTRIBUETED VIRTUALIZATION

Resource allocation in cloud computing has attracted the attention of the research community in the last few years. Armbrust, Srikantaiah and et al studied the problem of request scheduling for multi-tiered web applications in virtualized heterogeneous systems in order to minimize energy consumption while meeting performance requirements. They proposed a heuristic for a multidimensional packing problem as an algorithm for workload consolidation. In previous articles we have published two algorithms. Which were used to detect deadlock in resources allocation heterogeneous distributed platforms. We provide deadlock detection algorithms and resolve the optimization problems of resources based the recovery of resources allocated. In 2016, we provide deadlock detection algorithms and resolve

optimal problems according to groups of users. Most of the studies were set to study scheduling policy effectiveness in resources allocation. Sotomayor et al proposed a lease-based model and implemented First-Come-First-Serve (FCFS) scheduling algorithm and a greedy based VM mapping algorithm to map leases that include some of VMs with/without start time and user specified duration to a set of homogeneous physical machines (PMs). Much of the research conducted homogeneous physical machines (PMs). Not much research provide resources for heterogeneous systems. To maximize performance, these scheduling algorithms tend to choose free load servers when allocating new VMs. On the other hand, the greedy algorithm can allocate a small lease (e.g. with one VM) to a multi-core physical machine. In this study, we propose solutions to detect deadlock in resource supply, then proceed to build the resource supply automatically detects and prevents deadlock occurs. This issue is also effective in allocation resources.

Chương 2

SYSTEM MODEL RESOURCE ALLOCATION IN HETEROGENEOUS DISTRIBUTED PLATFORMS

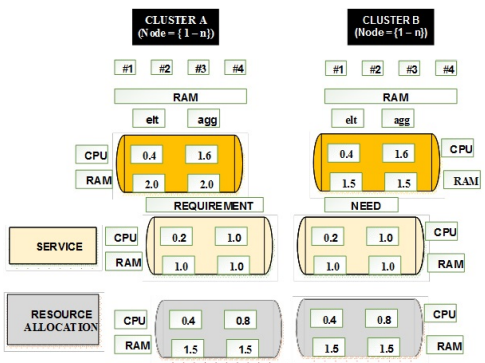
2.1. System Model Resource Allocation In Heterogeneous Distributed Platforms

We consider a service hosting platform composed of H heterogeneous hosts, or nodes. Each node comprises D types of different resource (*such as CPUs, network cards, hard drives, or system memory*). For each type of resource under consideration a node may have one or more distinct resource elements (*a single real CPU, hard drive, or memory bank*). Services are instantiated within virtual machines that provide analogous virtual elements. For some types of resources, like system memory or hard disk space, it is relatively easy to pool distinct elements together at the hypervisor or operating system level so that hosted virtual machines can effectively interact with only a single larger element. For other types of resources, like CPU cores, the situation is more complicated.

These resources can be partitioned arbitrarily among virtual elements, but they cannot be effectively pooled together to provide a single virtual element with a greater resource capacity than that of a physical element. For these types of resources, it is necessary to consider the maximum capacity allocated to individual virtual elements, as well as the aggregate allocation to all vital elements of the same

type. An allocation of resources to a virtual machine specifies the maximum amount of each individual element of each resource type that will be utilized, as well as the aggregate amount of each resource of each type. An allocation is thus represented by two vectors, a maximum elementary allocation vector and an aggregate allocation vector. Note that in a valid allocation it is not necessarily the case that each value in the second vector in an integer multiple of the corresponding value in the first vector, as resource demands may be unevenly distributed across virtual resource element. The distributed computation consists of a set processes, and processes only perform computation upon receiving one or more messages. Once initiated, the process continues with its local computation, sending and receiving additional messages to other processes, until it stops again. Once a process has stopped, it cannot spontaneously begin new computations until it receives a new message. The computation can be viewed as spreading or diffusing across the processes much like a fire spreading through a forest.

Figure 1 illustrates an example with two nodes and one service. Node A, B are comprised of 4 cores and a large memory. Its resource capacity vectors show that each core has elementary capacity 0.8 for an aggregate capacity of 3.2. Its memory has a capacity of 1.0, with no difference between elementary and aggregate values because the memory, unlike cores, can be partitioned arbitrarily. No single virtual CPU can run at the 0.9 CPU capacity on this node. The figure shows two resource allocations one on each node. On both nodes, the service can be allocated for memory it requires. Informally speaking, a deadlock is a system state where requests are



Hình 2.1 Example problem instance with two nodes and one service, showing possible resource allocations

waiting for resources held by other requesters which, in turn, are also waiting for some resources held by the previous requests. Wonly consider the case where requests

are processors on virtual machine resource allocation on heterogeneous distributed platforms. A deadlock situation results in permanently blocking a set of processors from doing any useful work. There are four necessary conditions which allow a system to deadlock [**?**]:

(a) *Non-Preemptive*: resources can only be released by the holding processor;

(b) *Mutual Exclusion*: resources can only be accessed by one processor at a time;

(c) *Blocked Waiting*: a processor is blocked until the resource becomes available.

(d) *Hold-and-Wait*: a processor is using resources and making new requests for other resources that the same time, without releasing held resources until some time after the new requests are granted.

2.2. The nVM-out-of-NPM model

A heterogeneous distributed platforms are composed of a set of an asynchronous processes (p_1, p_2,. . .,p_n) that communicates by message passing over the communication network. Based on the basic work of the authors Kshemkalyani-Singhal, and other works such as Menasce-Muntz, Gligor - Shattuck, Ho - Ramamoorthy, Obermarck, Chandy, and Choudhary. They have the same opinions that the requested resource model of distributed system is divided into five model resource requirements. It's simple resource models, resource models OR, AND resource models, models AND/OR, and model resource requirements P-out-of-Q. Through this model, the researchers have discovered a technical proposal deadlock corresponding to each model. In this work, we use model P-out-of-Q as a prerequisite for developing research models provide resources in the cloud. The nVM-out-of-NPM problem depicts on-demand resource allocation to n VMS residing in N servers, where each VM may use resources in more than one server concurrently. Thus, we model it to guide the design of algorithm prevents deadlock in resource allocation among VMs each of which may use the resource in various servers concurrently. First, we introduce the following notations:

E_{ijt} is the amount of resources allocated to VM_{ij} at time t, where

$$\sum_{i}^{N} E_{ij} = \sum_{i}^{N} (A_{ij} + \sum_{i=1}^{n} C_{\mathrm{ij}}) \tag{2.1}$$

Bảng 2.1 The description of notations using the formula

Notations	**Meanings**
E	E is the total CPU or other resource that are available to all VMs in N server.
n	n is the number of VMs residing in the N server.
EN_{ijt}	EN_{ijt} is the native resources allocated to VM_{ij} in server i.
$EO^{x}_{\mathrm{ij}t}$	$EO^{x}_{\mathrm{ij}t}$ is the amount of resources allocated to VM_{ij} in server x.
C_{ij}	C_{ij} is the minimum threshold of resources allocated to VM_{ij}.
D_{ijt}	D_{ijt} is the native resources demand of VM_{ij} at time t.
Φ_{ij}	Φ_{ij} is the tolerable quality threshold of the application hosted in VM_{ij}.
Q_{ijt}	Q_{ijt} is the quality of application hosted in VM_{ij} at time t.

E_{ijt} obeys the rules as follows:

$$\begin{aligned} &E \geq \sum_{i=1}^{N} \sum_{j=1}^{n_N} E_{\mathrm{ij}t} \\ &E_{\mathrm{ij}t} \geq C_{\mathrm{ij}} \geq 0 (i = 1, ..., N; j = 1, ..., nN) \end{aligned} \tag{2.2}$$

The resource allocation problem is how to control the resource allocation to VMs with the goal of minimizing the function F_t, giving the limited resources. We get the following formulation:

$$F_t = \sum_{i=1}^{N} \sum_{i=1}^{n_i} \frac{f_{\mathrm{ij}}(EN_{\mathrm{ij}t}, \sum_{x=1}^{n} EO^{x}_{\mathrm{ij}t}, D_{ijt})}{\Phi_{ij}} \times SP_i$$

$$\begin{cases} \sum_{i=1}^{N} \sum_{i=1}^{n_i} E_{ijt} \leq E \\ E_{\mathrm{ij}t} \geq C_{\mathrm{ij}} \quad (i = 1, 2, ..., n; j = 1, 2, ..., m) \\ \sum_{i=1}^{n_i} EN_{\mathrm{ij}t} + \sum_{j=1}^{n_i} EO^{i}_{\mathrm{ij}t} \leq E_i \\ E_{it} \geq C_{ij} \quad (i = 1, 2, ..., n; j = 1, 2, ..., m). \end{cases} \tag{2.3}$$

Chương 3

TECHNICAL SOLUTION FOR RESOURCE ALLOCATION IN HETEROGENEOUS DISTRIBUTED PLATFORMS

3.1. Deadlock Detection for Resource Allocation in Heterogeneous Distributed Platforms

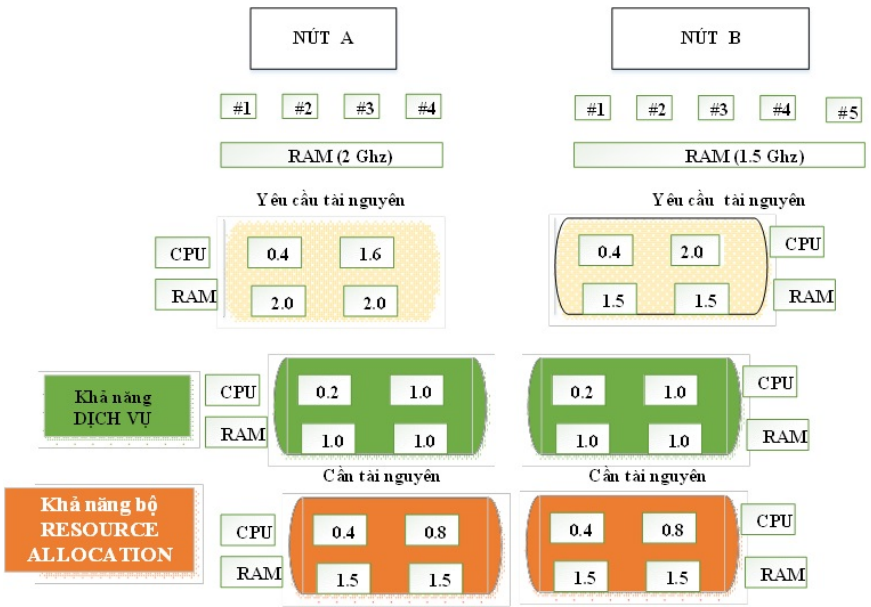
Deadlock detection can be represented by a *Resource Allocation Graph* (RAG), commonly used in operating systems and distributed systems. A RAG is defined as a graph (V, E) where V is a set of nodes and E is a set of ordered pairs or edges (v_i, v_j) such that $v_i, v_j \in V$. V is further divided into two disjoint subsets: $P = \{p_0, p_1, p_2, ..., p_m\}$ where P is a set of processor nodes shown as circles in Fig.3.6; and $Q = \{q_0, q_1, q_2, ..., q_n\}$ where Q is a set of resource nodes shown as boxes in Fig.3.6. A RAG is a graph bipartite in the P and Q sets. An edge $e_{ij} = (p_i, q_j)$ is a racist edge if and only if $p_i \in P$, $q_j \in Q$. The maximum number of edges in a RAG is $m \times n$. A node is a sink when a resource (*processor*) has only incoming edge(s) from processor(s) (resource(s)). A node is source when a resource (*processor*) has only out-going edge(s) to processor(s) (resource(s)). A path is a sequence of edges $\varepsilon = \{(p_{i1}, q_{j1}), (q_{j1}, p_{i2}), ..., (p_{ik}, q_{jk+1}), (q_{js}, p_{is+1})$ where $\varepsilon \in E$. If a path starts from and ends at the same node, then it is a cycle. A cycle does not contain any sink or source nodes.

The focus of this thesis is deadlock detection. For our virtual machine resource allocation for heterogeneous distributed platform deadlock detection implementation, we make three assumptions. First, each resource type has one unit. Thus, a cycle is a sufficient condition for deadlock. Second, satisfies request will be granted immediately, making the overall system expedient. Thus, a processor is blocked only if it cannot obtain the requests at the same time. All proposed algorithms, including those based on a RAG, have $O(m \times n)$ for the worst case.. We propose a deadlock detection algorithm with $O(min(m, n))$ based on a new matrix representation. The proposed virtual machine resource allocation on heterogeneous distributed platforms deadlock detection algorithm makes use of parallelism and can handle multiple requests/grants, making the pro-posed algorithm faster than the $O(1)$ algorithm.

3.2. Our Algorithm

On the use of graphs representing RAG matrix is presented, we approach me to propose a deadlock detection algorithm in heterogeneous platforms. The basic idea of this algorithm is reported to reduce the matrix by removing the corresponding columns or rows. This is continued until the matrix can not be reduced any more columns and rows. At this time, if the matrix still contains row (s) or column (s), which may also consider other factors, not anymore, then consider a cycle exists, it cannot declare published at least one deadlock in the system. If not, there is no deadlock. The description of our algorithm shows in the algorithm 1.

The following example illustrates how the algorithm works. In each iteration of this parallel algorithm, at least one reduction can be performed if the matrix is reducible. Hence, it takes at most $min(m, n)$ iterations to complete the deadlock detection. An example has two processors: VM_1 and VM_2, as p_1 and p_2 respectively. The devices are S_1, S_2, and S_3, as q_1, q_2 and q_3 respectively as shown in Fig.**??**.



Hình 3.1 Resource allocation on heterogeneous distributed platforms

The matrix representation of this example is shown in table. In this matrix, the first and second column contains both g and r, and hence is not reducible. However, the third column contains only g. Thus $m_{12} = g$ can be reduced. At the same time, each row is also examined, however, there is no reduction possible. Since there is one reduction, the next iteration will be carried out. In the second iteration, the first

Algorithm 1 Parallel Deadlock Detection Algorithm (PDDA Improved)

Input: $P_i^{j(CPU)^*}$, $P_i^{j(RAM)^*}$ from IaaS provider i;

Output: new resource $r_j^{CPU^{(n+1)}}$, $r_j^{RAM^{(n+1)}}$;

BEGIN

Calculate optimal resource allocation to provide VM:

$x_i^{j(CPU)^*}, x_i^{j(RAM)^*} = Max\{U_{IaaS}\}$;

Computes new resource:

If $(C_j^{CPU} \geq \sum_i x_i^{j(CPU)}, C_j^{RAM} \geq \sum_i x_i^{j(RAM)})$

{

$r_j^{CPU^{(n+1)}} = \max\{\varepsilon, r_j^{CPU^{(n)}} + n(\sum_i x_i^{j(CPU)} - C_j^{CPU})\}$

$r_j^{RAM^{(n+1)}} = \max\{\varepsilon, r_j^{RAM^{(n)}} + n(\sum_i x_i^{j(RAM)} - C_j^{RAM})\}$

Return new resource $r_j^{CPU^{(n+1)}}$, $r_j^{RAM^{(n+1)}}$;

}

Else

{

$M = [m_{ij}]^{m \times n}$, where $m_{ij} = \begin{cases} r \ if \ \exists \, (p_i, q_j) \in E. \\ g \ if \ \exists \, (p_i, q_j) \in E. \\ 0 \quad otherwise \end{cases}$, $(i = 1, \ldots, m; j = 1, \ldots, n)$

$\Lambda = \{m_{ij} | m_{ij} \in M, m_{ij} \neq 0\}$;

DO

$Reducible = 0$;

For each column:

If $((\exists m_{ij} \in \forall k, k \neq i, m_{kj} \in \{m_{ij}, 0\}))$

{

$\Lambda_{column} = \Lambda - \{m_{ij} | j = 1, 2, 3, ..., m\}$;

$reducible = 1$;

};

For each row:

If $((\exists m_{ij} \in \forall k, k \neq i, m_{kj} \in \{m_{ij}, 0\})$

{

$\Lambda_{row} = \Lambda - \{m_{ij} | j = 1, 2, 3, ..., m\}$;

$reducible = 1$

};

$\Lambda = \Lambda_{column} \cap \Lambda_{row}$

UNTIL $(reducible = 0)$;

}

Detect Deadlock

If $(\Lambda \neq 0)$

Deadlock;

Else

No deadlock; **END**;

and second columns still contain both g and r, and hence are not reducible. At the same time, each row is also checked, but no reduction is possible for any row. Since there are no more reductions, a conclusion is drawn. In this case, hardware deadlock detection takes two iterations and finds a deadlock.

Let us remove the edge (p_2, q_2) in this case and consider it again. In this matrix, the first column cannot be reduced, because of the existence of both g and r, while the second and third columns can be reduced, be-cause the second column has only one r and no g's, and the third column has only one g and no r's. At the same time, the first and second rows cannot be reduced, be-cause of the existence of both g and r in each row.

Since this iteration has a reduction, in the first step, we will be re-executed by the second and third columns having been removed. During the second iteration, the first column is not reduced, because there are both r and g in this column. However, the first row can be reduced because on r is in this row.

Then Step 1 is executed again in what is now a third iteration of the Parallel Deadlock Detection Algorithm. There are no more reductions, because the matrix now is empty. In the second step, we concludes that there is no deadlock. In this case, three iterations are taken to complete detections.

3.2.1. Experiments and results

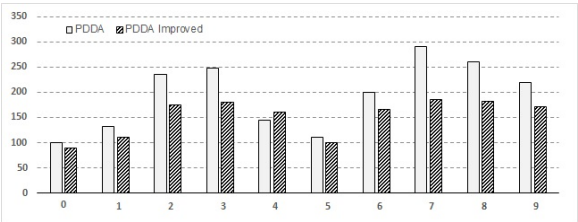
Cloud computing resource allocation method based on improving PDDA has been validated on CloudSim, the platform is an open source platform, we use the Java language to program algorithm implementation class.

The experiments give 7 tasks, by CloudSim's own optimization method and improved algorithm PDDA to run the 9 tasks, experiment data as follows Table.

The comparative analysis of experimental result can be seen in many times, apter task execution, although there were individual time improved PDDA algorithm response time was not significantly less than an optimal time algorithm, in most cases, improved algorithm is better than the optimal time algorithm, thus validated the correctness and effectiveness.

Bảng 3.1 Comparison the optimal time of our algorithm to PDDA algorithm

			PDDA			PDDA Improved			
Cloudlet ID	Data center ID	VM ID	Start	End	Time	Start	End	Time	Improved (%)
0	2	0	0.1	100.1	100	0.1	90.1	90	10.00%
5	2	0	0.1	110.1	110	0.1	100.1	100	9.09%
1	2	1	0.1	132.1	132	0.1	110.1	110	16.67%
4	2	1	0.1	145.1	145	0.1	160.1	160	10.34%
6	2	2	0.1	200.1	200	0.1	165.1	165	17.50%
9	2	2	0.1	220.1	220	0.1	172.1	172	21.82%
2	2	4	0.1	235.1	235	0.1	175.1	175	25.53%
3	2	4	0.1	248.1	248	0.1	180.1	180	27.42%
8	2	3	0.1	260.1	260	0.1	182.1	182	30.00%
7	2	3	0.1	290.1	290	0.1	185.1	185	36.21%



Hình 3.2 Comparison the optimal time of algorithms

3.3. A New Technical Solution for Resource Allocation in Heterogeneous Distributed Platforms

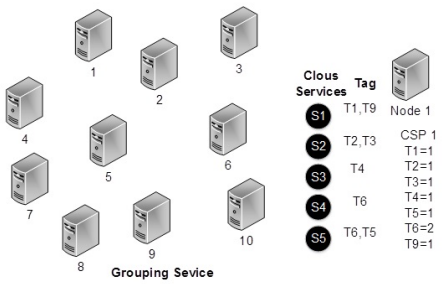
In this section, we will first introduce the matrix representation of a deadlock detection problem. The algorithm is based on this matrix representation. Next, we present some essential features of the proposed algorithm. This algorithm is, and thus can be mapped into a cloud architecture which can handle multiple requests/grants simultaneously and can detect multiple deadlocks in linear time, hence, significantly improving performance.

3.3.1. Group method for cloud service providers

The cloud service providers also support the Infrastructure as a Service (IaaS) where the components of the virtualized platform are extracted to different nodes within the group. Specifically, the platform base is accessed in a single node while its components can be on different nodes. A cloud user does not need to know where

the services are located within the nodes, but these are fertilized in one platform. Figure 1 shows this function by virtualizing the platform for the cloud user where the platform base is in node A while the services are in nodes B and C.

A cloud service provider can be independent in providing cloud services. However, there are times that the demands of services are very high. A frequently accessed cloud services from a cloud provider produces high loads. In queuing systems, requests from clients are queued if the system processor is busy with processing requests. The number of queued jobs and the processing time of each job represent the loads of the node. The queuing system can be improved by forwarding the queued task to an idle or less loaded node and this is only possible if there are additional resources that are bound to the node. The proposed cloud system supports the resource binding by grouping of cloud service providers to provide additional resources and prevent late responses from high frequent accessed cloud service.The grouping of cloud service provider is managed by the grouping service. Grouping the cloud service providers processes in cluster analysis where it groups data with similar property values. This approach assumes that the grouped cloud providers will have a fast response on serving the large number of requests by sharing the similar cloud services. Moreover, the collaboration among the cloud providers will be more efficient if related cloud services are grouped.



Hình 3.3 Collection of cloud service providers which is managed by the grouping service

Figure 3.3 illustrates the initial configuration of the cloud service providers before the process of grouping. Each cloud service provider is provided with a unique identification number. Every cloud service (S_n) has tag values and the count of tags are determined in the upper left of Figure 3.3. In the small box from the top right of Figure 3.3, a cloud service provider (CSP_1) shows its cloud services and property values. The cloud services for grouping, where i is the index tag, us the same cloud service used for the search of cloud services. All cloud service tags are incremented

to summarize the cloud service provider property to be processed in the cluster analysis. Their property values serve as data for the cluster analysis. Cluster analysis divides data into groups such that similar data objects belong to the same cluster and dissimilar data objects to different cluster. Partitioning methods construct c partition of data, where each partition represents a cluster and $c < k$ (c is the number of cluster and k is the number of data).

$$\min \sum J = \sum_{i=1}^{c} \left(\sum_{k=1, u_k \in C_i} m_{ik} \left\| u_k - cv_i \right\| \right) \tag{3.1}$$

The objective function shown in Eq.1 depends on the distances between vectors u_k and cluster centers cv_i where u_k is the data value and cv is the center value. The m_{ik} represents the group member of the $u_k \in \{0, 1\}$ and C_i represents the cluster index. The partitioned clusters are typically defined by a $c \times k$ binary characteristic matrix M, called the membership matrix, where each element m_{ik} is 1 if the k^{th} data point u_k belongs to cluster i, and 0 otherwise. The min $\sum J$ is the objective function within cluster i where i is the index of the cluster. The function min $\sum J$ in Eq.1 is to find the minimal value of the group to determine a more compact grouping. The J_i is minimized by several iterations and stops if either the improvement over the previous iteration is below a certain tolerance or J_i is below a certain threshold value.

3.3.2. Our Deadlock Detection Algorithm

On this basis of the matrix representation, we propose a deadlock detection algorithm. The basic idea of this algorithm iteratively reduces the matrix by removing those columns or rows corresponding to any of the following cases:

- A row or column of all 0's
- A source (a row with one or more r's (no g's), or a column with one g and no r's)
- A sink (a row with one or more g's (no r's), or a column with one r's but no g's)

This continues until the matrix cannot be reduced any more. At this time, if the matrix still contains row(s) or column(s) in which there are non-zero elements, then there is at least one deadlock. Otherwise, there is no deadlock. The description of our algorithms shows belows.

Algorithm 2 Our Deadlock Detection Algorithm

BEGIN

1. Place c point in the space represented by the objects that are being clustered.
   These points represent initial group center values (cv).
2. Assign each object (u_k) to the group that has the closest center value.
3. When all objects have been assigned, recalculate the positions of the c center value.
4. Initialization
   $M = [m_{ik}]^{m \times n}$, $m_{ik} \in \{0, 1\}$ where $m_{ik} = \begin{cases} 1 \; if \; \exists \, (u_i, c_k) \in E. \\ 0 \quad otherwise \end{cases}$, $(i = 1, \ldots, m; k = 1, \ldots, n)$
   $\Lambda = \{m_{ik} | m_{ik} \in M, m_{ik} \neq 0\}$;
5. Remove all sink and sources
   DO
   $Reducible = 0$;
   For each column:
   If $((\exists m_{ik} \in \forall j, j \neq i, m_{jk} \in \{m_{ik}, 0\}))$
   {
   $\Lambda_{column} = \Lambda - \{m_{ik} | j = 1, 2, 3, ..., m\}$;
   $reducible = 1$;
   };
   For each row:
   If $((\exists m_{ik} \in \forall j, j \neq i, m_{ik} \in \{m_{ik}, 0\})$
   {
   $\Lambda_{row} = \Lambda - \{m_{ik} | j = 1, 2, 3, ..., m\}$;
   $reducible = 1$
   };
   $\Lambda = \Lambda_{column} \cap \Lambda_{row}$
   UNTIL $(reducible = 0)$;
   }
6. Detect Deadlock
   If $(\Lambda \neq 0)$ Deadlock;
   Else No deadlock;

END;

The following example illustrates how the algorithm works. In each iteration of this algorithm, at least one reduction can be performed if the matrix is reducible. Hence, it takes at most $min(m, n)$ iterations to complete the deadlock detection. The cloud service providers are described by the cloud service tags (t) and these are used as inputs for the cluster analysis. These cloud service tags are summarized by a vector value (u_k) which is used for the objective function in Eq.1. the cv is properties of all cloud services in the cloud system which are used in calculating the Euclidean distance in Eq.1.

In this section, a sample data is provided to each cloud service provider and the cluster analysis is processed to group the cloud service providers into 3 groups. Table.3.2 shows the property values from the cloud service providers from Figure 3.3. The tags from T_1 to T_{10} are used in this data.

Bảng 3.2 Property values of each cloud service provider (CPS)

	Properties (*count of tags*)									
CSP ID	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	2	0	0	1	0
2	1	0	1	1	1	2	0	0	0	0
3	1	1	1	1	1	2	0	0	1	0
4	0	0	0	0	0	0	2	2	1	1
5	0	0	0	0	0	0	2	1	1	1
6	0	0	0	0	0	0	2	2	1	1
7	1	1	1	1	1	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

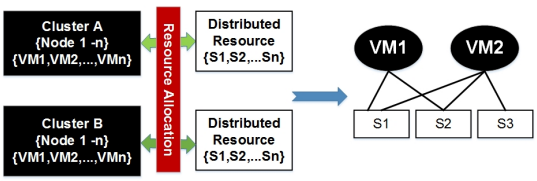
Table.3.2 shows the values of cloud service provider properties based on counting the tags from the cloud services it hosts. Each node is defined by these values represent an input object for the cluster analysis. Also, a property values as zero if tags were not found in the cloud service providers.

Example 2: *Cluster analysis procedure.*

- Calculate the properties of CSP_1 to $Group_1$.
  - $|1-1|+|1-1|+|1-1|+|1-1|+|1-1|+|2-1|+|0-1|+|0-0|+|1-0|+|0-0| = 4$
  - $|1-0|+|1-0|+|1-0|+|1-0|+|1-0|+|2-0|+|0-0|+|0-1|+|1-1|+|0-1| = 14$
  - $|1-0|+|1-0|+|1-0|+|1-0|+|1-0|+|2-0|+|0-0|+|0-0|+|0-0|+|1-0| = 13$
- Choosing $Group_1$ for CSP_1.
- Calculate the mean of all properties in each group $Compactness(J) = 8.6$.
- Grouping:

– $Group_1 : 1, 2, 3, 7$; $Group_2 : 8, 9, 10$; $Group_3 : 4, 5, 6$

All property values of a cloud service provider represent the u_k in the Eq.1. The data center value will adjust every time there is a change in the members of the group and the calculation continues to minimize the objective function. The process of cluster analysis is executed by J is minimized. Each cloud service provider will be assigned to a specific group represented by A, B, C. After the classification in of the group, the cloud service providers to form a virtual group where the grouping service informs each cloud service provider its group assignment which is illustrated Figure 3.



Hình 3.4 Formation of virtual group after the cluster analysis

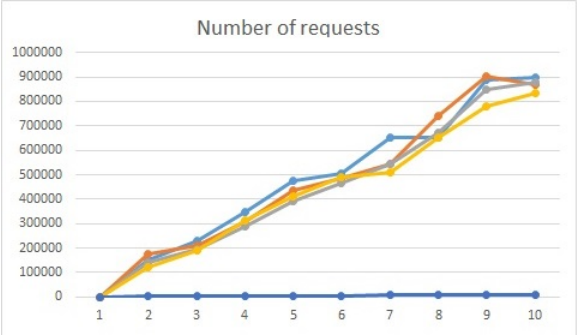
3.3.3. Simulation results and analysis

The method resource allocation based on improving DDA has been validated on CloudSim, the platform is an open source platform, we use the Java language to program algorithm implementation class. The experiments give 10 tasks, by CloudSim's own optimization method and improved algorithm DDA to run the 10 tasks, experiment data as follows:

- The comparative analysis of experimental result can be seen in many times, apter task execution, although there were individual time our improved DDA algorithm response time was not significantly less than an optimal time algorithm, in most cases, our improved algorithm is better than the optimal time algorithm, thus validated the correctness and effectiveness.

- The first case generated the request using a normal distribution of arrival time. This determines the performance of the algorithms to handle the arrival of tasks in an exponentially increasing number of requests. There were up 10000 requests generated and also these requests were assigned at random.

Theorem 1. Our algorithm detects deadlock if and only there exists a cycle in the state.



Hình 3.5 Improved DDA algorithm and optimal time algorithm comparison and analysis

Proof: Consider matrix M_{ik}

(a) Algorithm returns, by construction, an irreducible matrix $M_{i,k+j}$.

(b) By the definition of irreducible $M_{i,k+j}$ has no terminal edges, yielding 2 cases:

(i) $M_{i,k+j}$ is completely reduced, or

(ii) $M_{i,k+j}$ is incompletely reduced.

In case (i), if a system state can be completely reduced, then it does not have a deadlock. If a system state cannot be completely reduced, then the system contains at least one cycle. Given system heterogeneous platforms has a cycle is a necessary and sufficient condition for deadlock.

Theorem 2: In a RAG, an upper bound on the number of edges in a path is $2 \times min(m, n)$, where m is the number of resources and n is the number of processes.

Proof: Let us consider the following three possibilities:

(i) In case $m = n$, one longest path is $\{p_1, q_2, p_2, q_2, \ldots, p_n, q_m\}$ since this path uses all the nodes in the state system, and since every node in a path must be distinct (*i,e., every node can only be listed once*). The number of edges involved in the path is $2 \times m - 1$.

(ii) In case $m > n$ $(m - n > 0)$, one longest path is $\{q_1, p_1, q_2, p_2, \ldots, q_n, p_n, q_{n+1}\}$; this path cannot be lengthened since every node in a path must be distinct, and since all n process nodes are already used in the path. Therefore, the number of edges in this path is $2 \times n$.

(iii) In case $m < n$ $(m - n < 0)$, the number of edges involved in any longest part is $2 \times m$.

As a result, case (i), (ii) and (iii) show that the number of edges of the maximum possible longest path in a RAG state is $2 \times min(m, n)$. Algorithm when implemented in heterogeneous platform, completes its computation in at most $2 \times min(m, n) - 3 = O(min(m, n))$ steps, where m is the number of resources and n is the number of processes. When all the nodes in the smallest possible cycle are used, the longest path has three edges in this smallest possible cycle. Therefore, in the worst case, $2 \times min(m, n) - 3$ is an upper bound on the number of edges in the longest possible path that are not also part of a cycle. Hence, the number of iterations required to reach an irreducible state becomes at most $2 \times min(m, n) - 3 = O(min(m, n))$, the worst case.

3.4. Deadlock Prevention for Resource Allocation in Heterogeneous Distributed Platforms

3.4.1. The *n* VM-out-of-1PM model

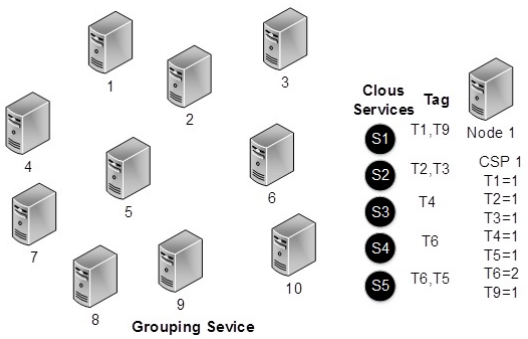
A heterogeneous distributed platforms are composed of a set of an asynchronous processes (p_1, p_2,. . .,p_n) that communicates by message passing over the communication network . Based on the basic work of the authors Kshemkalyani-Singhal, and other works such as Menasce-Muntz, Gligor - Shattuck, Ho - Ramamoorthy, Obermarck, Chandy, and Choudhary. They have the same opinions that the requested resource model of distributed system is divided into five model resource requirements. It's simple resource models, resource models OR, AND resource models, models AND/OR, and model resource requirements P-out-of-Q. Through this model, the researchers have discovered a technical proposal deadlock corresponding to each model. In this work, we use model P-out-of-Q as a prerequisite for developing research models provide resources in the cloud. The n VM-out-of-1PM problem depicts on-demand resource allocation to n VMS residing in N servers, where each VM may use resources in more than one server concurrently. Thus, we model it to guide the design of algorithm prevents deadlock in resource allocation among VMs each of which may use the resource in various servers concurrently. E_{ijt} is the amount of resources allocated to VM_{ij} at time t, where

$$E^{CPU} = A^{CPU} + \sum_{i=1}^{n} \sum_{j=1}^{m} C_{\mathrm{ij}}^{CPU} \tag{3.2}$$

The resource allocation problem is how to control the resource allocation to VMs with the goal of minimizing the function F_t, giving the limited resources. We get the following formulation:

$$\begin{aligned} &\min F_t = \sum_{i=1}^{n} \frac{f_i(E_{it}, D_{it})}{\Phi_i} \times SP_i \\ &\begin{cases} \sum_{i=1}^{n} E_{it} \leq E(i = (1, 2, ..., n) \\ E_{it} \geq C_{\text{ij}}(i = 1, 2, ..., n; j = 1, 2, ..., m) \end{cases} \end{aligned} \tag{3.3}$$

We can use methods to prevent deadlock to solve optimal resource model provides n VM-out-of-1PM. Our algorithm is based on wait-for graphs (WFG) algorithm is presented in section 4.



Hình 3.6 Example: A Resource Allocation System

3.4.2. Our Algorithm

In this thesis, we will approach proposed algorithm new for deadlock prevention maintains property of n-vertex directed graph when the new is added in the graph using two-way search. The time bound for the incremental cycle algorithm for deadlock prevention take O ($\sqrt{m}$) time bound for the m edge insertion in the directed graph. It reports the cycle when the algorithm detects for edge (v,m) that there exist a path from vertex w to v.

3.4.3. Experiments and results

We implement the designed deadlock prevention algorithm on a physical machine server with an Intel E5-2603V3 processor and 16G memory. Algorithm resource requirements and algorithms prevent deadlock in resource supply. Based on

Algorithm 1 Requests Resources Allocaition Algorithm (RRAA)

Input: $P_i^{j(CPU)^*}$, $P_i^{j(RAM)^*}$ from IaaS provider i;

Output: new resource $r_j^{CPU^{(n+1)}}$, $r_j^{RAM^{(n+1)}}$;

BEGIN

Operation request resource (r_i) in the critical section is

$csstate_i \longleftarrow$ trying;

$lrd_i \longleftarrow clock_i + 1$;

for each $j \in R_i$ do

if ($usedby_i$[j] =0) the send request (lrd_i,i) to p_j end for;

$sentto_i$ [j] $\longleftarrow$ true;

$usedby_i$[j] $\longleftarrow$ R

else $sentto_i$[j] $\longleftarrow$false

end if

end for;

$usedby_i$[i] $\leftarrow k_i$;

$wait(\sum_{j=1}^{n} usedby_i[j] \leq 1PM)$;

$csstate_i \longleftarrow$ in;

Operation release resource (r_i) in the critical section is

$csstate_i \longleftarrow$ out;

for each $j \in permdelayed_i$ do send permission(i,j) to p_j end for;

$R_i \leftarrow permdelayed_i$;

$permdelayed_i \leftarrow \oslash$

END.

the resource model provides n VM-out-of-1 PM. Methods of optimizing the use of functions in the formula 3. Optimal recovery method in materials allocated because the process still holds resources when finishing requirements. Data concerning the use of CPU, RAM and HDD were collected from the above physical machine servers at Data Center in every 4 hour during a period of 1 days. For comparison, the other server equipped with the same configuration have prevention deadlock algorithm,

Algorithm 2 Prevention Deadlock Algorithm (PDA)

Input: $P_i^{j(CPU)^*}$, $P_i^{j(RAM)^*}$ from IaaS provider i;

Output: new resource $r_j^{CPU^{(n+1)}}$, $r_j^{RAM^{(n+1)}}$;

BEGIN

When REQUEST(k,j) is received from p_j *do*

$clock_i \leftarrow \max(clock_i, \text{n})$;

$prio_i \leftarrow (\text{csstate}_i = \text{in}) \vee ((\text{csstate}_i = \text{trying}) \wedge ((\text{lrd}_i, \text{i}) < (\text{n,j})))$;

if ($prio_i$) then send NOTUSED(1PM) to p_j

else if($n_i \neq$ 1PM) then send NOTUSED(1PM - n_i) to p_j end if

$permdelayed_i \leftarrow permdelayed_i \cup \text{j}$

end if.

When permission(i,j) is received from p_j *do*

$1PM_i \leftarrow 1PM_i \setminus \text{j}$;

When NOTUSED(x) is received from p_j do $usedby_i[\text{j}] \leftarrow usedby_i[\text{j}]$ -x;

if $((csstate_i = trying) \wedge (usedby_i[j] = 0) \wedge (notsentto_i[j])$

then send REQUEST(lrd_i,i) to p_j

$sentto_i[\text{j}] \leftarrow$ true;

$usedby_i[\text{j}] \leftarrow$ 1PM;

end if.

END.

named native.The data were separated into two parts.

The first part was the set of data collected within the Algorithm 1 used for model n VM-out-of-1PM and the second part was the remaining data collected within the Algorithm 2 for testing. Compare data set obtained table we noticed that prevention deadlock results also brought confidence and the ability to bring greater efficiency. For simulation we need a special toolkit named CloudSim. It is basically a Library for Simulation of Cloud Computing Scenarios. It has some features such as it support for modeling and simulation of large scale Cloud Computing infrastructure, including data centers on a single physical computing node. It provides basic class for describing data centers, virtual machines, applications, users, computational re-

sources, and policies.

CloudSim supports VM Scheduling at two levels: First, at the host level where it is possible to specify how much of the overall processing power of each core in a host will be assigned at each VM. And the second, at the VM level, where the VMs assign specific amount of the available processing power to the individual task units that are hosted within its execution engine.

We saw infrastructure of Cloud, in which Data Center consist of different Hosts and the Host manages the VM Scheduler and VMs.

Cloud-let Scheduler determines how the available CPU resources of virtual machine are divided among Cloud lets. There are two types of policies are offered: Space - Shared (Cloud-let Scheduler Space Shared): To assign specific CPU cores to specific VMs.

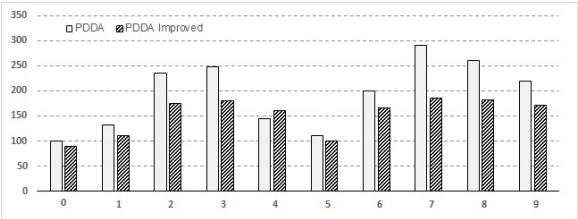
VM Scheduler determines how many processing cores of a host are allocated to virtual machines and how many processing cores will be delegated to each VM. It also determine how much of the processing core's capacity will effectively be attributed for given VM. Time-Shared (Cloud-let Scheduler Time Shared): To dynamically distribute the capacity of a core among VMs, test data are as follows:

Bảng 3.3 Requests Resources Allocation Algorithm (RRAA)

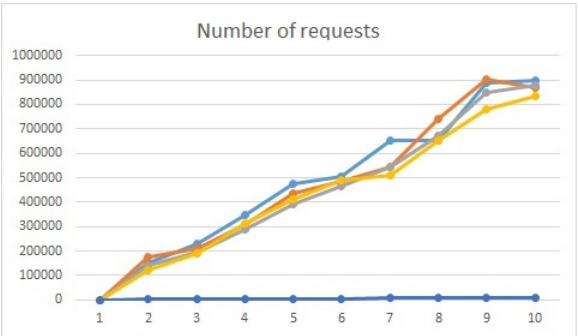
Method	Cloudlet ID	PM ID	VM ID	Start time	End time	Finish time (%)
0	1	1	0	0.1	100	22.22%
1	2	2	0	0.1	110	20.00%
2	3	2	1	0.1	132	27.27%
3	4	3	1	0.1	145	43.75%
4	5	3	2	0.1	200	39.39%
5	6	4	2	0.1	220	36.05%
6	7	4	3	0.1	235	42.86%
7	8	1	3	0.1	248	44.44%
8	9	2	4	0.1	260	50.55%
9	10	3	4	0.1	290	64.86%

The comparative analysis of experimental result can be seen in many times, apter task execution, although there were individual time improved PDA algorithm

response time was not significantly less than an optimal time algorithm, in most cases, improved algorithm is better than the optimal time algorithm, thus validated the correctness and effectiveness. The averaged resource usage of different execution strategies are reported in Figure 4.



Hình 3.7 Comparative analysis of experimental result algorithm RRAA with PDA



Hình 3.8 Comparative analysis of experimental result PDDA Improved with PDA

More exactly, the averaged CPU, Memory and Network utilization rates are 20 % . Due to the barrier synchronization, CPU and network utilization rate fluctuate dramatically in each graph processing super step. In contrast,the memory utilization rate is quite stable, as most memory are used to store the graph structure. At the end of execution,the memory usage reduces slightly as the number of active verticals reduced. Similar to above observations, instance resources are not fully utilized in all execution strategies. Generally, the higher instance capability,the higher CPU usage and network traffic; and the more instances, the lower CPU usage and network traffic. In the case of 16 instances, the CPU usage increases proportionally to the number of CPU cores, which indicates that the graph vertex programs are run by CPU cores in parallel.

3.4.4. Conclusion and Future Works

A prevention deadlock algorithm is implemented for resource allocation on heterogeneous distributed platforms. The prevent deadlock algorithm has O ($\sqrt{m}$) time

complexity, an improvement of approximate orders of magnitude in practical cases. In this way, programmers can quickly prevent deadlock and then resolve the situation, e.g., by releasing held resources. We focus on the application of the algorithm to prevent deadlock, when the process enters the critical section. From here we proceed to reschedule delivery system resources in a physical server. The application of the algorithms prevent this impasse in the proposed offer based resources distributed heterogeneous virtual machines require. Through this study, we found that the application of algorithms for preventing deadlock would best performance and take advantage of the physical resources.

CONCLUSION

A deadlock detection algorithm is implemented for resource allocation on heterogeneous distributed platforms. The deadlock detection algorithm has $O(min(m, n))$ time complexity, an improvement of approximately orders of magnitude in practical cases. In this way, programmers can quickly detect deadlock and then resolve the situation, e.g., by releasing held resources.

The proposed virtualization mechanism is used by the cloud service providers to group the appropriate cloud service providers and to distribute the cloud services within the number of the group. In providing a scalable sharing of resources, a group method for cloud service providers is supported in the proposed cloud system. Moreover, the task distribution from the resource management layer IaaS is improved by performing the proposed service assignment. The service assign is the main algorithm for the virtualization mechanism which improves the resource allocation and task distribution throughout the hardware resource.

Through this research, we found that the application of appropriate scheduling algorithms would give optimal performance to distributed resources of virtual server systems.