A technical solution for resource allocation in heterogeneous distributed platforms

Ha Huy Cuong Nguyen and Van Son Le

Department of Information Technology, Quangnam University, Viet Nam
Department of Information Technology, Da Nang University of Education,Viet Nam

Summary
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. There are more general types of resource allocation problems than those we consider here. In this paper, we present an approach for improving the deadlock detection algorithm, to schedule the policies of resource supply for resource allocation on heterogeneous. The deadlock detection algorithm has a run time complexity of O (min (m, n)), where m is the number of resources and n is the number of processes. We propose the algorithm for allocating multiple resources to competing services running in virtual machines on a heterogeneous distributed platform. The experiments also compare the performance of the proposed approach with other related work.

Key words:
Cloud Computing. Resource Allocation. Heterogeneous Distributed Platforms. Deadlock Detection

1. Introduction

Recently, there has been a dramatic increase in the popularity of cloud computing systems that rent computing resources on-demand, bill on a pay-as-you-go basis, and multiplex many users on the same physical infrastructure [1,2]. Cloud infrastructure services, also known as "Infrastructure as a Service (IaaS)", delivers computer infrastructure typically a platform virtualization environment as a service, e.g., Amazon EC2, Rackspace Cloud Servers. Rather than purchasing servers, software, data center space or network equipment, clients instead buy such services on a utility computing basis and the amount of resource consumed will typically reflect the level of activity. IaaS evolved from virtual private server offerings.

The increasing use of virtual machine technology in the data center, both leading to and reinforced by recent innovations in the private sector aimed at providing low-maintenance cloud computing services, has driven research into developing algorithms for automatic instance placement and resource allocation on virtualized platforms [1,2], including our own previous work. Most of this research has assumed a platform consisting of homogeneous nodes connected by a cluster. However,
there is a need for algorithms that are applicable to heterogeneous platform.

Heterogeneity happens when collections of homogeneous resources, formerly under different administrative domains, are federated and lead to a set of resources that belong to one of several classes. This is the case when federating multiple clusters at one or more geographic locations, e.g., grid computing, sky computing.

In this work we propose virtual machine placement and resource allocation deadlock detection algorithms that, unlike previous proposed algorithms, are applicable to virtualized platforms that comprise heterogeneous physical resources. More specifically, our contributions are:

We provide an algorithmic approach to detect deadlock and resource allocation issues in the virtualization platform heterogeneity. This algorithm is in fact more general, even for heterogeneous platforms, and only allowed to allocate minimal resources to meet QoS arbitrary force.

Using this algorithm, we extend previously proposed algorithms for the heterogeneous case.

We evaluate these algorithms via extensive simulation experiments, using statistical distributions of application resource requirements based on a real-world dataset provided by Google.

Most resource allocation algorithms rely on estimates regarding the resource needed for virtual machine instances, and do not refer to the issue of detecting and preventing deadlock. We studied the impact of estimation error and propose different approaches to mitigate these errors, and identify a strategy that works well empirically.

2. Related works

Resource allocation in cloud computing has attracted the attention of the research community in the last few years. Srikantaiah et al. [8] studied the problem of request scheduling for multi-tiered web applications in virtualized heterogeneous systems in order to minimize energy

Manuscript received January 5, 2016
Manuscript revised January 20, 2016

consumption while meeting performance requirements. They proposed a heuristic for a multidimensional packing problem as an algorithm for workload consolidation. Garg et al. [10] proposed near optimal scheduling policies that consider a number of energy efficiency factors, which change across different data centers depending on their location, architectural design, and management system. Warneke et al. [11] discussed the challenges and opportunities for efficient data processing in a cloud environment and presented a data processing framework to exploit the dynamic resource provisioning offered by IaaS clouds. Wu et al. [12] proposes a resource allocation for SaaS providers who want to minimize infrastructure cost and SLA violations. Addis et al. [13] proposed resource allocation policies for the management of multi-tier virtualized cloud systems with the aim to maximize the profits associated with multiple – class SLAs. A heuristic solution based on a local search that also provides availability, guarantees that running applications have developed. Abdelsalem et al. [14] created a mathematical model for power management in a cloud computing environment that primarily serves clients with interactive applications such as web services. The mathematical model computes the optimal number of servers and the frequencies at which they should run. Yazir et al. [15] introduced a new approach for dynamic autonomous resource management in computing clouds. Their approach consists of a distributed architecture of NAs that perform resource configurations using MCDA with the PROMETHEE method. Our previous works mainly dealt with resource allocation, QoS optimization in the cloud computing environment.

There are more general types of resource allocation problems than those we consider here. For instance:

1. We consider the possibility that users might be willing to accept alternative combinations of resources. For example, a user might request elementary capacity CPU, RAM, HDD rather than a specific.
2. We consider the possibility that resources might be shared. In this case, some sharing is typically permitted; for example, two transactions that need only to read an object can be allowed concurrent access to the object.
3. We begin by defining our generalized resource allocation problem, including the deadlock problem as an interesting special case. We then give several typical solutions.

3. Existing models and problem definitions

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) [16,17,18].

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.

Example 1 Resource allocation on heterogeneous distributed platforms

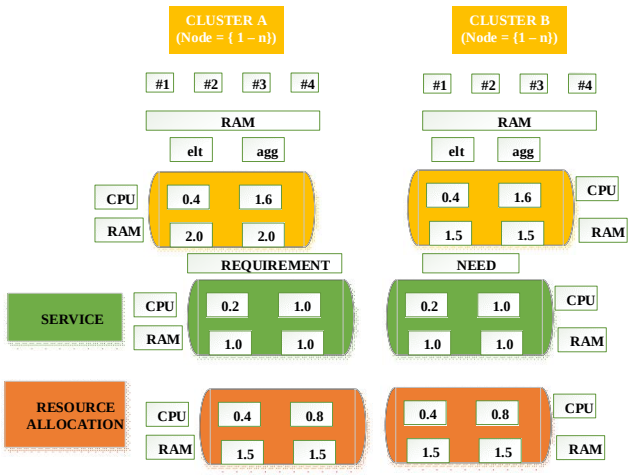

Fig. 1 Example problem instance with two nodes and one service, showing possible resource allocations.

Informally speaking, a deadlock is a system state where requests are waiting for resources held by other requesters which, in turn, are also waiting for some resources held by the previous requests. In this paper, we only 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[3]: (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; and (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.

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 (vi,vj) such that vi,vj $\in$ 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 Figure 1; and $Q = \{q_0, q_1, q_2, ..., q_n\}$ where Q is a set of resource nodes shown as boxes in Figure 1. A RAG is a graph bipartite in the P and Q sets. An edge eij=(pi,qj) is a request edge if and only if pi $\in$ P, qj $\in$ 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 outgoing 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 paper is deadlock detection. For our virtual machine resource allocation on heterogeneous distributed platforms deadlock detection implementation, we make three assumptions. First, each resource type has one unit. Thus, a cycle is a sufficient condition for deadlock [3]. Second, satisfies request will be granted immediately, making the overall system expedient [3]. 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.. In this paper, 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 ism and can handle multiple requests/grants, making the proposed algorithm faster than the O(1) algorithm[16,17].

4. A 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.

4.1 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. Fig 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 [26,27], 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[44]

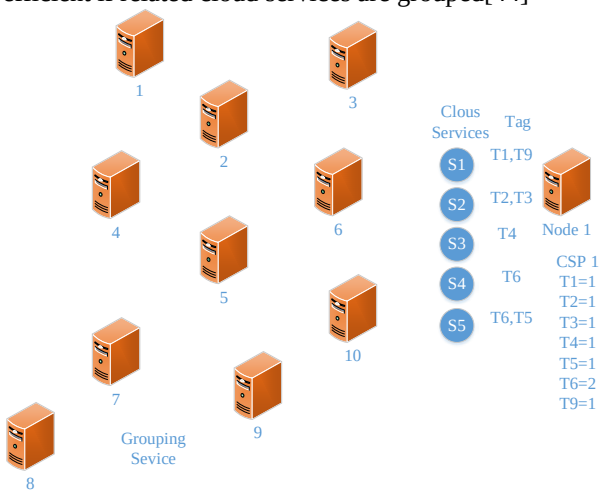

Fig. 2 Collection of cloud service providers which is managed by the grouping service

Figure 2 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 (sn) has tag values and the count of tags are determined in the upper left of Figure 2. In the small box from the top right of Figure 2, a cloud service provider (CSP1) 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 in Section 3. 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[48]. 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.

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

The objective function shown in Equation 1 depends on the distances between vectors uk and cluster centers cvi where uk is the data value and cv is the center value. The mik represents the group member of the uk, which is {0,1} and Ci 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 mik is 1 if the kth data point uk 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 Equation 1 is to find the minimal value of the group to determine a more compact grouping. The Ji is minimized by several iterations and stops if either the improvement over the previous iteration is below a certain tolerance or Ji is below a certain threshold value.
The algorithm for deadlock detection is composed of the following steps:

4.2 A 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 but no g's, or a column with one g and no r's);
A sink (a row with one or more g's but 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 this algorithms shows in the.
Algorithm. Deadlock Detection Algorithm

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. Step 4: Initialization $M = [m_{ik}]^{m \times n}$,

Where $m_{ik} \in \{1,0\}$, (i =1, …,m and k =1,…,n)

m_{ik} = 1 if $\exists (u_i,c_i) \in E$.

m_{ik} = 0, otherwise.

$$\Lambda = \{m_{ik} \mid m_{ik} \in M, m_{ik} \neq 0\};$$

5. Step 5: Remove all sink and sources

DO {

Reducible = 0;

For each column:

$$if\ (\exists m_{ik} \in \forall j, j \neq i, m_{jk} \in \{m_{ik}, 0\})\{$$
$$\Lambda_{column} = \Lambda - \{m_{ik} \mid j = 1,2,3,\ldots,m\},$$
$$reducible = 1;$$
$$\}else\{\}$$

For each row:

$$if\ (\exists m_{ik} \in \forall j, j \neq i, m_{jk} \in \{m_{ik}, 0\})\{$$
$$\Lambda_{row} = \Lambda - \{m_{ik} \mid j = 1,2,3,\ldots,m\},$$
$$reducible = 1;$$
$$\}else\{\}$$
$$\Lambda = \Lambda_{column} \cap \Lambda_{row};$$
$$\}UNTIL(reducible = 0);$$

6. Step 6: Detect Deadlock

If ($\Lambda \neq 0$), then deadlock exits.

If ( ), then no deadlock exits.

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 (uk) which is used for the objective function in Equation 1. the cv is properties of all cloud services in the cloud system which are used in calculating the Euclidean distance in Equation 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 4 shows the property values from the cloud service providers from Fig 2. The tags from T1 to T10 are used in this data.

Table 1: Property values of each cloud service provider (CPS)

CSP ID	Properties (count of tags)									
	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 1 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 to represent an in put object for the cluster analysis. Also, a property is valued as zero if tags were not found in the cloud service providers.

Example 2: Cluster analysis procedure

Calcutate 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

Group1: 1,2,3,7

Group2: 8,9,10

Group3: 4,5,6

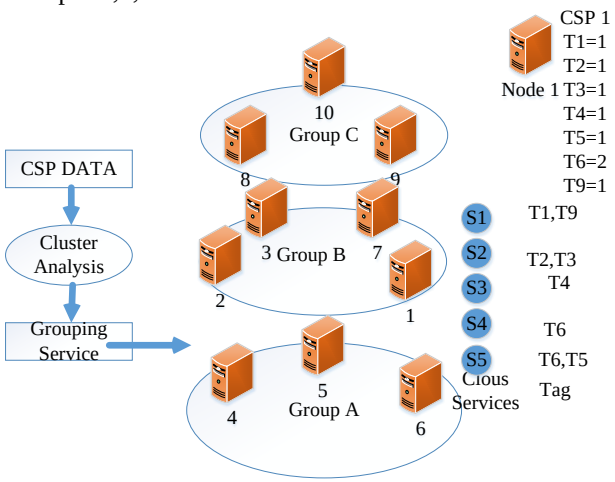

Fig. 1. Formation of virtual group after the cluster analysis

All property values of a cloud service provider represent the uk in the Equation 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 Fig 3.

4.3 Proof of the correctness of PDDA

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

Proof: Consider matrix Mik (a) Algorithm returns, by construction, an irreducible matrix Mi, k+j. (b) By the

definition of irreducible Mi,k+j has no terminal edges, yielding two cases: (i) Mi,k+j is completely reduced, or (ii) Mi,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.

4.4 Proof of the Run-time Complexity PDDA

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) m = n, (ii) m >n, or (iii) m<n. In case (i), where m equals n, one longest path is {p1, q2, p2,q2,…,pn,qm} 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). In this case, the number of edges involved in the path is $2 \times m-1$. In case (ii), where m is greater than n (i.e,. $m - n > 0$), one longest path is {q1,p1,q2,p2,…,qn,pn,qn+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 $2 \times n$. Likewise, for case (iii), where n is greater than m (i.e., n – m >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.

5. Simulation results and analysis

In this paper, cloud computing resource allocation method 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 improved DDA 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 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 request generated and also these requests were assigned in randomly.

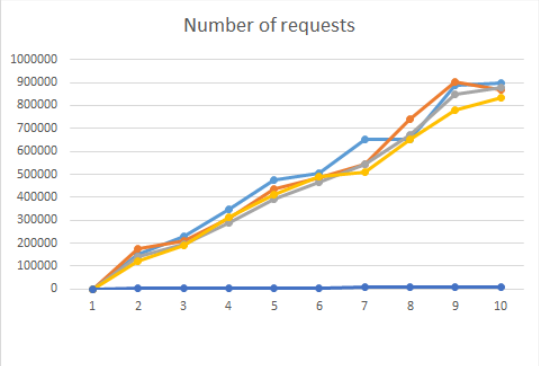

Fig. 3 Improved DDA algorithm and optimal time algorithm comparison and analysis

6. Conclusion

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. In this paper, the service assign is the main algorithm for the virtualization mechanism which improves the resource allocation and task distribution throughout the hardware resource. A deadlock detection algorithm is implemented for resource allocation on heterogeneous 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.

Our main approach focuses on applying deadlock detection algorithms for each type of lease contracts and applying the proposed algorithm in resource allocation on heterogeneous distributed platform.

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

References

[1] Ha Huy Cuong Nguyen, Van Son Le, Thanh Thuy Nguyen, "Algorithmic approach to deadlock detection for resource allocation in heterogeneous platforms", Proceedings of 2014 International Conference on Smart Computing 3 - 5 November, Hong Kong, China, page 97 – 103.

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D.,Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commune. ACM53(4), 50–58 (2010)

[3] M. Andreolini, S. Casolari, M. Colajanni, and M. Missouri, "Dynamic load management of virtual machines in a cloud architectures," inCLOUDCOMP, 2009.

[4] P. Shiu, Y. Tan and V. Mooney "A novel deadlock detection algorithm and architecture", Proc. CODES 01, pp.73 -78, (2001).

[5] Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commune. Rev. 39(1), 50–55 (2009)

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D.,Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing.Technical Report No. UCB EECS-2009-28, University of California at Berkley, USA, Feb 10, 2009

[7] Kaur P.D., Chana I.: Enhancing Grid Resource Scheduling Algorithms for Cloud Environments. HPAGC 2011, pp. 140–144, (2011)

[8] Vouk, M.A.: Cloud computing: Issues, research and implementations. In: Information Technology Interfaces. ITI 2008. 30th International Conference on, 2008, pp. 31–40, (2008)

[9] Srikantaiah, S., Kansal, A., Zhao, F.: Energy aware consolidation for cloud computing. Cluster Comput. 12, 1–15 (2009)

[10] Berl, A., Gelenbe, E., di Girolamo, M., Giuliani, G., de Meer, H., Pentikousis, K., Dang, M.Q.:Energy-efficient cloud computing. Comput. J. 53(7), 1045–1051 (2010)

[11] Garg, S.K., Yeo, C.S., Anandasivam, A., Buyya, R.: Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. J Distrib Comput. Elsevier Press,Amsterdam, (2011)

[12] Warneke, D., Kao, O.: Exploiting dynamic resource allocation for efficient data processing in the cloud. IEEE Trans. Distrib. Syst. 22(6), 985–997 (2011).

[13] Wu, L., Garg, S.K., Buyya, R.: SLA-based Resource Allocation for a Software as a Service Provider in Cloud Computing Environments. In: Proceedings of the 11th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2011), Los Angeles, USA, May 23–26, (2011)

[14] Addis, B., Ardagna, D., Panicucci, B.: Autonomic Management of Cloud Service Centers with Availability Guarantees. 2010 IEEE 3rd International Conference on Cloud Computing, pp 220–207,(2010)

[15] Abdelsalam, H.S., Maly, K., Kaminsky, D.: Analysis of Energy Efficiency in Clouds. 2009 Computation, World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns,pp. 416–422, (2009)

[16] Yazir, Y.O., Matthews, C., Farahbod, R.: Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis. IEEE 3rd International Conference on Cloud Computing, pp. 91–98, (2010)

[17] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," JPDC, vol. 70, no. 9, pp. 962–974, (2010).

[18] S. Benin, A. I. Bucur, and D. H. Epema, "A measurement-based simulation study of processor co-allocation in multi-cluster systems," inJSSPP, pp. 184–204, (2003).

[19] A. Buttery, J. Kurzak, and J. Dongarra, "Limitations of the PlayStation 3 for high performance cluster computing," U Tenn., Knoxville ICL, Tech. Rep. UT-CS-07-597, (2007).

[20] S. Banen, A. I. Bucur, and D. H. Epema, "A measurement-based simulation study of processor co-allocation in multi-cluster systems," inJSSPP, pp. 184–204, (2003).

[21] A. Buttari, J. Kurzak, and J. Dongarra, "Limitations of the PlayStation 3 for high performance cluster computing," U Tenn., Knoxville ICL, Tech. Rep. UT-CS-07-597, (2007).

[22] D. P. Mitchell and M. J. Merritt, "A distributed algorithm for deadlock detection and resolution," in Proc.ACM Symposium on Principles of Distributed Computing, pp. 282–284,1984.

[23] Ajay D. Kshemkalyani, Mukesh Singhal. "Distributed Computing Principles, Algorithms, and Systems", Cambridge University Press, 2008.

[24] Bondy JA, Murty USR (2008) Graph theory. Springer graduate texts in mathematics. Springer,Berlin. ISBN 978-1-84628-970-5.

[25] Fournier JC (2009) Graph theory and applications. Wiley, New York. ISBN 978-1-848321-070-7.

[26] Greg N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM Journal on Computing, v.16 n.6, p.1004-1022.

Ha Huy Cuong Nguyen received the M.S degrees in Danang University in 2010. From 2011 until now, he studied at the center DATIC, University of Science and Technology - The University of DA Nang. At the center of this research, he doctoral thesis "Studies deadlock prevention solutions in resource allocation for distributed virtual systems".