ISSN 2047-3338

Detection and Avoidance Deadlock for Resource Allocation in Heterogeneous Distributed Platforms

Ha Huy Cuong Nguyen[1] and Van Son Le[2]

[1]Department of Information Technology, QuangNam University, Tam Ky City, Viet Nam
[2]Department of Information Technology, Da Nang University of Education, Da Nang, Viet Nam
[1]nguyenhahuycuong@gmail.com, [2]levansupham2004@yahoo.com

***Abstract*— In a heterogeneous distributed platforms large problem is distribute among the processors (such as among the computers of distributed computing system or among the computers of parallel computing system etc.) to make it cost effective and less time consuming instead of a computer system with single processor. There are more general types of resource allocation problems than those we consider here. In this paper, we present an approach for improving detection and avoidance algorithm, to schedule the policies of resource supply for resource allocation on heterogeneous. We propose an algorithm for allocating multiple resources to competing services running in virtual machines platforms.**

***Index Terms*— Cloud Computing, Resource Allocation, Heterogeneous Distributed Platforms, Deadlock Detection and Avoidance Deadlock**

I. 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. These cloud computing environments provide an illusion of infinite computing resources to cloud users that they can increase or decrease their resources. In many cases, the need for these resources only exists in a very short period of time.

The increasing use of virtual machine technology in data centers, 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 platforms.

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 geographical 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 to 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 deadlocks. We studied the impact of estimation errors and propose different approaches to mitigate these errors, and identify a strategy that works well empirically.

II. 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 consumption while meeting performance requirements. They proposed a heuristic for a multidimensional been 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 cloud environment and presented a data processing framework to exploit the dynamic resource provisioning offered by IaaS clouds. Wu et al. [12] propose 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 for 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 user 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 detection problem as an interesting special case. We then give several typical solutions.

III. System Model Resource Allocation in Heterogeneous Distributed Platforms

Resource allocation in cloud computing has attracted the attention of the research community in the last few years. Cloud computing presents a different resource allocation paradigm than either grids or batch schedulers [2]. In particular, Amazon C2 [10], is equipped to, handle may smaller computer resource allocations, rather than a few, large request as is 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.

Like traditional distributed system before we can see a heterogeneous distributed system consists of a set of processes that are connected by a communication network. The communication delay is finite but unpredictable [21], [22].

A). The Application

A heterogeneous distributed program is composed of a set of n asynchronous processes $p_1, p_2, \ldots, p_n$ that communicates by message passing over the communication network. We assume that each process is running on a different processor. The processor does not share a common global memory and communicate solely by passing messages over the communication network. There is no physical global clock in the system to which processes have instaneous access. The communication medium may deliver messages out of order, messages may be lost garble or duplicated due to timeout and retransmission, processors may fail and communication links may go down. The system can be modeled as a directed graph in which vertices represent the processes and edge represent unidirectional communication channels.

We use the platform graph, for the grid platform. We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected graph. See an example in Fig. 1 with 8 processors and 11 communication links. Each node is a computing resource (a processor, or a cluster, or node).

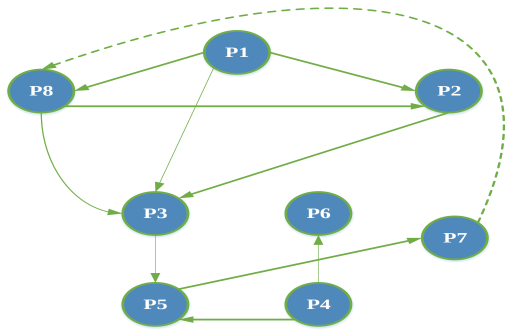


Fig. 1. An example simple platform

A process can be in two states: running or blocked. In the running state (also called active state), a process has all the needed re and is either executing or is ready for execution. In the blocked state, a process is waiting to acquire some resource.

B). The Architecture

The target heterogeneous platform is represented by a directed graph, the platform graph. There are p nodes P_1, $P_2, \ldots, P_n$ that represent the processors. In the example of figure 1 there at eight processors, hence n = 8.

Each edge represents a physical interconnection. Each edge e_{ij}: $P_i \rightarrow P_j$ is labeled by value $c_{i,j}$ which represents the time to transfer a message of unit length between P_i and P_j, in either direction: we assume that the link between P_i and P_j is bidirectional and symmetric. A variant would be to assume two unidirectional links, one in each direction, with possibly different label values. If there is no communication link between P_i and P_j we let $c_{i,j} = +\infty$, so that $c_{i,j} < +\infty$ means that P_i and P_j are neighbors in the communication graph.

C). Wait – For – Graph (WFG)

In distributed systems, the sate of the system can be modeled by directed graph, called a wait for graph (WFG) [21] – [25]. In a WFG, nodes are processors and there is a directed edge from node P_1 to mode P_2 if P_1 is blocked and is waiting for P_2 to release some resource. A system is deadlocked if and only if there exists a directed cycle or knot in the WFG.

Let us first of all describe the deadlock condition problem more precisely.

A set S = $\{s_1, s_2,\ldots s_k\} \subseteq \mathcal{E}$ of k > 1 entities is deadlocked when the following two conditions simultaneously hold:

Each entity $s_i \in S$ is waiting for an event permission that must be generated from another entity in the set;

No entity $s_i \in S$ can generate a permission while it is waiting.

If these two conditions hold, the entities in the set will be waiting forever, regardless of the nature of the permission and of why they are waiting for the "permission"; for example, it could be because s_i needs a resource held by s_j in order to complete its computation.

A useful way to understand the situations in which deadlock may occur is to describe the status of the entities during a computation, with respect to their waiting for some events, by means of a directed graph $\overrightarrow{W}$, called wait-for graph.

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 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 $e_{ij}=(p_i,q_j)$ is a request 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 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 $O(m\times n)$ in the worst case.. In this paper, we propose 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 $O(m\times n)$ algorithm [16], [17].

IV. Deadlock Detection 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.

A). Matrix Representation of a Deadlock Detection Problem

In graph theory, any directed graph can be represented with an adjacency matrix [3]. Thus, we can represent a RAG with an adjacency matrix. However, there are two kinds of edges in a RAG: grant edges, which point from resources to processors, and request edges, which point from processors to resources. To distinguish different edges, we designate elements in the adjacency matrix with three different values as shown in Figure 2. This Figure shows the matrix representation of a given system with processors $p_1, p_2,\ldots,p_i,...,p_m$ and resources $q_1, q_2,\ldots,q_j,\ldots,q_n$. The leftmost column is the processors label column. The top row is the resources label row. If there is a request edge (p_i,q_j) in the RAG, corresponding element in the matrix is r. If there is a grant edge (q_i,p_j) in the RAG. The corresponding element in the matrix is g. Otherwise, the value of the element is 0.

This variant of the adjacency matrix of a RAG (V,E) can be defined formally as follows:

$M=[m_{ij}]^{m\times n}$, $(1 \le i \le m,\ 1 \le j \le n)$, where m is the number of processors and n is the number of resources.

$m_{ij} \in \{r,g,0\}$

$m_{ij} = r$, if f $\exists(p_i, q_j) \in E$

$m_{ij} = g$, if f $\exists(p_i, q_j) \in E$

$m_{ij} = 0$, if otherwise

This matrix provides a template able to represent request and grant combinations. Note that each resource has at most one grant, that is, there is at most one g in a column at any time. However, there is no constraint on the number of requests from each processor.

If there are deadlocks in a system, there must be at least one cycle in its RAG, that is, there must be a sequence of edges, $\varepsilon=\{(p_{i_1}, q_{j_1}),(q_{j_1}, p_{i_2}),...,(p_{i_k}, q_{j_k}),(q_{j_k}, p_{i_{k+1}}),...,(p_{i_s}, q_{j_s}),(q_{j_s}, p_{i_1})\}$,where $\varepsilon \subseteq E$. In the matrix representation, this cycle is mapped into a sequence of matrix elements $\omega=\{m_{i_1 j_1}, m_{i_2 j_1},..., m_{i_k j_k}, m_{i_{k+1} j_k}, m_{i_s j_s}, m_{i_1 j_s}\}$ where are requests(r's) and $m_{i_2 j_1}, m_{i_3 j_2},..., m_{i_{k+1} j_k}, ...m_{i_1 j_s}$ are grants (g's). By this fact, we can detect deadlocks in a system with its adjacency matrix. Next, we will present the new detection algorithm.

B). Deadlock Detection Algorithm

On this basis of the matrix representation, we propose a deadlock detection algorithm. The basic idea in this algorithm is iteratively reducing 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 method is shown in algorithm.

```
Algorithm:  Deadlock Detection Algorithm
    Input: $P_i^{j(\mathrm{CPU})^*}$ ; $P_i^{j(\mathrm{RAM})^*}$ from IaaS provider i;
    Step 1: calculate optimal resource allocation to provide VM. $x_i^{j(\mathrm{CPU})^*}, x_i^{j(\mathrm{RAM})^*} = Max\{U_{IaaS}\}$;
    Step 2: Computes new resource
    If $C_j^{CPU} \geq \sum_i x_i^{j(\mathrm{CPU})}, C_j^{RAM} \geq \sum_i x_i^{j(RAM)}$ then
    $r_j^{CPU^{(n+1)}} = \max\{\varepsilon, r_j^{CPU^{(n)}} + n(\sum_i x_i^{j(CPU)} - C_j^{CPU})\}$;
    $r_j^{RAM^{(n+1)}} = \max\{\varepsilon, r_j^{RAM^{(n)}} + n(\sum_i x_i^{j(\mathrm{RAM})} - C_j^{RAM})\}$;
        Return new resource $r_j^{CPU^{(n+1)}}$ ; $r_j^{RAM^{(n+1)}}$
    Else
    Step 3: Initialization
     $M = [m_{ij}]^{m \times n}$,
    Where $m_{ij} \in \{r,g,0\}$, $(i = 1, \ldots, m$ and $j = 1, \ldots, n)$
     $m_{ij} = r$ if $\exists\, (p_i,q_j) \in E$.
     $m_{ij} = g$ if $\exists\, (p_i,q_j) \in E$.
     $m_{ij} = 0$, otherwise.
     $\Lambda = \{m_{ij} \mid m_{ij} \in M, m_{ij} \neq 0\}$;
    Step 4: Remove all sink and sources
    DO {
        Reducible = 0;
        For each column:
            $if\, (\exists m_{ij} \in \forall k, k \neq i, m_{kj} \in \{m_{ij}, 0\})\{$
            $\Lambda_{column} = \Lambda - \{m_{ij} \mid j = 1,2,3,\ldots,m\}$,
            $reducible = 1$;
            $\}else\{\}$
        For each row:
            $if\, (\exists m_{ij} \in \forall k, k \neq i, m_{kj} \in \{m_{ij}, 0\})\{$
            $\Lambda_{row} = \Lambda - \{m_{ij} \mid j = 1,2,3,\ldots,m\}$,
            $reducible = 1$;
            $\}else\{\}$
            $\Lambda = \Lambda_{column} \cap \Lambda_{row}$;
            $\}UNTIL(reducible = 0)$;
    Step 5: Detect Deadlock
        If ( $\Lambda \neq 0$ ), then return deadlock exits.
        If ( $\Lambda = 0$ ), then return no deadlock exits.
    Output: new resource $r_j^{CPU^{(n+1)}}$ ; $r_j^{RAM^{(n+1)}}$
```

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

C). Our Algorithm to Avoidance Deadlock

In heterogeneous distributed platforms if there are n processes for each processor and m resources the algorithm will be as follow:

Step 1: In any of the processes of a processor using one of the resources in a specific time period (i.e., the period-1) then no other requirements for the resources will be allocated for the specified time period.

Step 2: The resource occupied can be used by other processes within a specific time period (i.e., the period-2) only after the release by the process of step 1 by approximately time-1. During the period-2 has no other process will be able to use that resource.

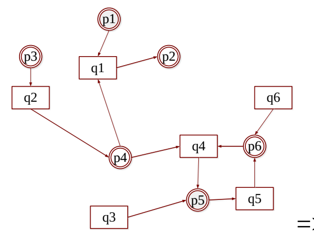
Step 3: In the same way, a process of the processor is determined in step 2 can use an O resources in a specific time period (i.e., the period-3) in the absence the other will be able to use that resource.

Step 4: The resources that were occupied in step 3 can be used by other processes defined in step 1 in a specific time period (i.e., the period-4) only after released by the process of step 3 by approximately-3. During the period-4 has no other process will be able to use that resource.

D). Proof of the Correctness of DDA

Example 2 State matrix representation

The system in state shown in Fig. 2 (a) can be representation in the matrix form show in (b). For the sake of better understanding, we will describe it in the matrix representation, shown in Fig. 3 (c) from now on.

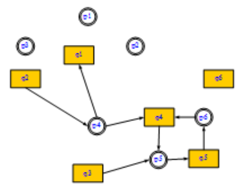


$$\Rightarrow M_{ij} = \begin{bmatrix} r & g & 0 & r & 0 & 0 \\ 0 & 0 & r & g & 0 & 0 \\ 0 & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & r & g & r \\ 0 & 0 & 0 & 0 & r & g \\ 0 & 0 & 0 & 0 & 0 & r \end{bmatrix}$$

=>

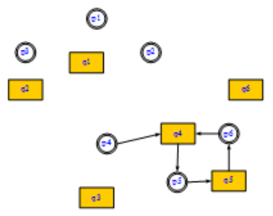
M_{ij}	P1	P2	P3	P4	P5	P6
Q1	r	g		r		
Q2			r	g		
Q3					g	
Q4				r	g	r
Q5					r	g
Q6						r

Fig. 2. Matrix representation example



M_{ij}	P1	P2	P3	P4	P5	P6
Q1						
Q2			r	g		
Q3						
Q4				r	g	r
Q5					r	g
Q6						

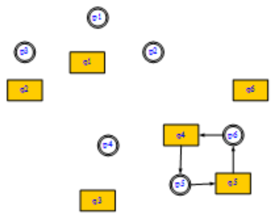
Step 1 (a)



M_{ij}	P1	P2	P3	P4	P5	P6
Q1						
Q2						
Q3						
Q4				r	g	r
Q5					r	g
Q6						

Step 2 (b)

I



M_{ij}	P1	P2	P3	P4	P5	P6
Q1						
Q2						
Q3						
Q4					g	r
Q5					r	g
Q6						

Step 3 (c)

Fig. 3. A sample sequence of reduction steps and deadlock detection

V. Simulation Results And Analysis

In this paper, resource allocation method based on improved DDA has been validated on CloudSim, the platform is an open source platform, we used Java language to program algorithm implementation class. Experiments give *n* tasks, by CloudSim's own optimization method and improved algorithm.

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 request. There were up 100 request generated and also these requests were assigned in randomly.

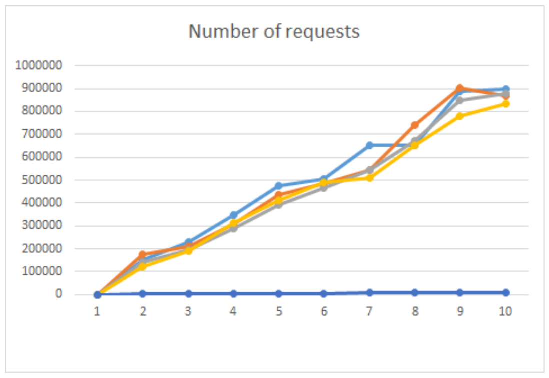


Fig. 4. Message overheads by network latencies using the generated requests based on arrival time in normal distribution

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

VI. Conclusion

Deadlock is a highly unfavorable situation that can occur in any multiprocessor system. The occurrence of a deadlock can cripple parts of a multiprocessor system. It is necessary to develop control policies that avoid deadlocks by restricting the freedom in resource allocation. In this paper we have presented a fast deadlock detection and deadlock avoidance methodology that is easily applicable to heterogeneous distributed platforms. Once a deadlock is detected, it must be somehow resolved. In a heterogeneous distributed platforms large problem is distribute among the processors (such as among the computers of distributed computing system or among the computers of parallel computing system etc.) to make it cost effective and less time consuming instead of a computer system with single processor. But since the large program is divided into smaller programs chances of occurring deadlock becomes high. So, it is very necessary to apply deadlock avoidance method in multiprocessor system. In this paper we have also provided the algorithm which simple and easy to understand. We can also implement this method everywhere in our computational life as needed.

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, pp. 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. Commun. ACM53(4), 50–58 (2010)

[3]. M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in cloud architectures," in CLOUDCOMP, 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. Commun. 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 Com-putation 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. Banen, A. I. Bucur, and D. H. Epema, "A measurement-based simulation study of processor co-allocation in multi-cluster systems," in JSSPP, pp. 184–204, (2003).

[19]. 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).

[20]. 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.

[21]. A.D.Kshemkalyani, and M.Singhal. (1999), A One-Phase Algorithm to Detect Distributed Deadlocks in Replicated Databases, IEEE Trans. Knowledge and Data Eng., vol. 11, No. 6, pp. 880-895.

[22]. RajkumarBuyya, Chee Shin Yeo, and Srikumar Venugopal, Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities, International Conference on High Performance Computing, 2008.

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

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

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