Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 17 (2013) 1162 – 1169

Procedia Computer Science

Information Technology and Quantitative Management (ITQM2013)

A task scheduling algorithm based on QoS-driven in Cloud Computing

Xiaonian Wu[a,*], Mengqing Deng[a], Runlian Zhang[a], Bing Zeng[b], Shengyuan Zhou[a]

[a]*School of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China*
[b]*Science and Technology on Communication Security Laboratory, Chengdu 610041, China*

Abstract

Quality of Service is an inevitable issue needing to be deal with in task scheduling of cloud computing. This paper proposes a task scheduling algorithm based on QoS-driven for cloud computing. Firstly, in order to reflect the precedence relation of tasks, the proposed algorithm computes the priority of tasks according to the special attributes of tasks, and then sorts tasks by priority. Secondly, the algorithm evaluates the completion time of each task on different services, and schedules each task onto a service which can complete the task as soon as possible according to the sorted task queue. The experimental results based on CloudSim show that the algorithm can achieve good performance and load balancing by QoS driving from both priority and completion time.

Selection and peer-review under responsibility of the organizers of the 2013 International Conference on Information Technology and Quantitative Management

Keywords: Cloud computing; Task scheduling; Quality of service; Priority

1. Introduction

Cloud computing is a new mode of business computing, and it promises to provide reliable services through "Cloud" centers built on virtualized computing and storage technologies [1]. From a "Cloud" anywhere in the world, users can gain computing power, storage and a variety of software services by using a variety of applications, and pay based on what they use. It is a key issue how to dispatch efficiently and reasonably tasks of users to different resources according to the Quality of Service (QoS) requirements of both Cloud Computing Center and users, which belongs to task scheduling.

However, the scheduling of applications for parallel and distributed computing systems is one of the most challenging NP-complete problems [2]. Currently the NP-completeness of the scheduling problem has inspired researchers to propose numerous heuristic algorithms, such as Min-Min, GA, SA, A*, etc [3]. Many improved algorithms [4, 5] based on existed algorithms are proposed to meet the requirement of environments.

* Corresponding author. Tel.: +86-13077658295
E-mail address: xnwu@guet.edu.cn.

Selection and peer-review under responsibility of the organizers of the 2013 International Conference on Information Technology and Quantitative Management
doi:10.1016/j.procs.2013.05.148

QoS is the collective effort of service performance, which determines the degree of satisfaction of a user for the service [6]. Commonly, QoS is expressed by the following qualitative measures, such as completion time, latency, execution price, packet loss rate, throughput and reliability.

In order to meet the QoS of cloud computing, the paper [7] proposed a job scheduling algorithm based on Berger model. The algorithm establishes dual fairness constraint. Firstly it classifies tasks by QoS preferences, and establishes the general expectation function in accordance with the classification of tasks. Secondly it defines resource fairness justice function to judge the fairness of the resources allocation. According to constraints, the algorithm always assigns tasks onto the optimal resources in order to satisfy the QoS requirements of users. In paper [8], a reliability-driven scheduling architecture was designed to meet task dependency applications' reliability requirements. A reliability priority rank (RRank) was introduced to estimate task's priority by considering reliability overheads. Furthermore, based on directed acyclic graph (DAG), a scheduling algorithm with duplication for precedence constrained tasks was proposed to achieve reliability for applications. Paper [9] presented slot selection algorithms in economic models for independent job batch scheduling in distributed computing with non-dedicated resources. It divided the scheduling of the job batch into two phases. With an eye to efficiency and economic policy, a scheduling scheme that features multi-variant search was built. In order to maximize resource utilization and acceptance of leases for the infrastructure as a Service (IaaS) clouds, paper [10] provided a scheduling algorithm to support deadline sensitive leases in Haizea to minimize request rejection rate while satisfying resource allocation policies offered by cloud provider.

This paper proposes a task scheduling algorithm based on QoS-driven in Cloud Computing (TS-QoS). The TS-QoS computes the task priority according to attributes of task such as user privilege, the priority expectation to be scheduled, task lengthen and the pending time of task in queue. Then tasks are sorted by the priority. And in the scheduling process, each task is assigned onto the service with the minimum completed time over all available services.

The rest of the paper is organized as follows: Section 2 gives related works. Section 3 gives detailed description of the TS-QoS. Section 4 describes the simulation experiment and experiment analysis. Section 5 gives the conclusions.

2. Task scheduling Model

In order to adapt to the parallel and distributed cloud computing environment and deal with QoS scheduling, we adopt the dynamic batching mode. In this mode, tasks are not mapped onto services as they arrive; instead they are collected in a set that is examined for mapping. And according to the collected information including the requirements of all tasks and the real-time state of all services, we can design more reasonable scheduling strategy to deal with QoS.

In this paper, we deal with QoS scheduling based on following strategies: (1) the task owned higher priority should be scheduled prior to tasks with lower priority; (2) a task should be completed as soon as possible.

2.1. The service description

In cloud computing, resources are encapsulated into services. Supposing a service set is denoted as $Service(m)$, then, $Service(m) = \{S_1, S_2, S_3, \ldots, S_j, \ldots, S_m\}$. Where, $m \in N$, N is a natural number, and $S_j \in Service(m)$, $j \in [1, m]$. Then, we define a service S_j by a multi-tuple as follows:

$S_j = \{SID, SProvider, SServicename, SAbility\}$

Where, each attribute of service S_j is described as follows:

(1) *SID* is ID of S_j;

(2) *SProvider* is the provider of S_j;

(3) *SServicename* exposes name of service and method of service, etc;

(4) *SAbility* is service ability of S_j, which mainly includes computing power, communication bandwidth and storage capacity. We set *SAbility* = {*processor*, *bandwidth*, *storage*}.

2.2. The task description

Task is medium or container that expresses or holds requirements of users. Generally task has been classified into the independent meta-task and the dependent task. Here, we only discuss meta-task. Supposing a meta-task set is denoted as *Task*(*n*), then, *Task*(*n*) = {T_1 , T_2 , T_3 ,…, T_i,…, T_n}. Where, $n \in N$, N is a natural number, and $T_i \in Task(n)$, $i \in [1, n]$. Then, we define a meta-task T_i by a multi-tuple as follows:

T_i={*TID, TUserType, TServType, TPriorExp, TAbilReq, TL, TArriTime, TState, TData*}

Next is a detailed account of meta-task T_i.

(1) *TID* is ID of meta-task T_i;

(2) *TUserType* stands by the privilege class of owner of T_i. Referring to the commercial mode, we set *TUserType* ∈ {*A, B, C*};

(3) *TServType* denotes the expected services of task T_i;

(4) *TPriorExp* is the expected scheduled priority of T_i, which shows the urgency of task T_i to be scheduled. Expected scheduled priority includes urgent priority, high priority, middle priority and low priority. So, we set *TPriorExp* ∈ {*urgent*, *high*, *mid*, *low*};

(5) *TAbilReq* denotes the service ability expectation to execute task T_i. Responding to the service definition, we set *TAbilReq* = {*processor*, *bandwidth*, *storage*};

(6) *TL* is the task length of task T_i. Here we assume *TL* is the number of instructions that the loop unrolled contained in task T_i. And it shows the computing workload of the task;

(7) *TArriTime* is the submitted time of task T_i;

(8) *TState* is the current state of task T_i, including idle, suspending, mapping, executing and completed;

(9) *TData* is the input data and output data, which would be stored into files.

From the definition of task, we can see that tasks maybe focus on computing, or communication, or storage, or their combination. In next section, we only discuss the computing task.

2.3. The priority computation of tasks

The priority is a comprehensive concept. Here, we estimate and calculate priority of tasks through user privileges, the urgency that tasks are scheduled, and the latency time queuing up and the task workload.

According to the task description, the user privilege corresponds to *TUserType*, the urgency that tasks are scheduled matches *TPriorExp*. Here, we just discuss the computing workload about task workload, so we denote the task workload as *TL*. And the latency time (*LT*) queuing up can be calculated by the current time and *TArriTime*. After introducing *LT* into priority of the task, the priority can change dynamically and increase continuously. This can solve “starving to death” issue, and follow the “first come first served” principle.

The priority is a hybrid concept which involves many attributes of task. In order to merge attributes into priority, we apply a normalization concept. Normalization can speed up data convergence by converting and mapping data within a certain range, and convert a dimensional data into a non-dimensional data. And it can dispel the adverse effect on the hybrid priority because of a wide difference among values of different attributes. Here, we use linear conversion, the formula is as follows.

$$Y_i = (x_i - MinValue)/(MaxValue - MinValue) \quad (1)$$

where， Y_i is the normalization result of special attribute of task T_i; x_i is the value of special attribute of task T_i; *MaxValue* and *MinValue* are the maximum and minimum value of special attribute from all tasks waiting to be scheduled respectively.

According to the above normalized attributes of tasks, we calculate the priority of tasks as follows.

$$P(i) = \alpha \times NTU_i + \beta \times NTP_i + \gamma \times NTL_i + \omega \times NLT_i \quad (2)$$

Where, $P(i)$ is the priority of task T_i. $\alpha, \beta, \gamma, \omega \in [0,1]$ are weights of priority, and $\alpha + \beta + \gamma + \omega = 1$. Of course, it would have different impact on task priority by adjusting weights $\alpha, \beta, \gamma, \omega$. NTU_i, NTP_i NTL_i, NLT_i are the normalization for *TUserType*, *TPriorExp*, *TL*, *LT* of task T_i respectively.

2.4. The service selection for completion time

It can reduce the completion time of tasks to complete tasks as soon as possible. Of course, to reduce the completion time of a task would shorten latency of the remainder tasks in queue. In order to carry out the strategy, we need to compute the completion time of each task on different services in the scheduling process. And some terminologies about the completion time are described as follows [10].

Definition 1. The expected execution time of task T_i is the amount of time taken by service S_j to execute T_i given that S_j has no load when T_i is assigned.

Definition 2. The expected time to computed (*ETC*) matrix is a matrix that is stored on the scheduling system where the mapping is stored, and contains the estimates for the expected execution times of all tasks on all services. In an *ETC* matrix, the elements along a row indicate the estimates of the expected execution times of a given task on different services, and those along a column give the estimates of the expected execution times of different tasks on a given service. Where the *entry*(*i,j*) is the expected execution time of task T_i on service S_j, and we can denotes it as *ETC*(*i,j*).

Definition 3. The start time (*ST*) matrix is a matrix that is stored on the scheduling system, and it contains the estimates for the earliest time that can be used for services after these services have executed and completed tasks allocated on. Where the *entry*(*j*) is the start time of service S_j, and we can denotes it as *ST*(*j*).

Definition 4. The minimum completion time (*MCT*) matrix is a matrix that is stored on the scheduling system where the mapping is stored, and contains the estimates for the expected completion times of all tasks on all services. In an *MCT* matrix, the elements along a row indicate the estimates of the expected completion times of a given task on different services, and those along a column give the estimates of the expected completion times of different tasks on a given service. Where the *entry*(*i,j*) is the expected completion time of task T_i on service S_j, and we can denotes it as *MCT*(*i,j*). And the formulation of *MCT*(*i,j*) is as follows:

$$MCT(i, j) = ETC(i, j) + ST(j) \quad (3)$$

From the above terminologies, we can see that the element *MCT*(*i,j*) in an *MCT* matrix records the total expected completion times of different tasks on a given service S_j. In fact, the total completion time of each service is the computing load of the service. That is to say, the greater the value for the element *MCT*(*i,j*), the heavier the load of service S_j. Load balancing is a very important metric of task scheduling, which impacts on fairness and efficiency of the system.

To schedule each task onto a service which has the minimum value of *MCT*(*i,j*), in that way, tasks would be completed as soon as possible. That is to say, it can help to reduce the completion time of tasks and achieve well load balancing.

2.5. The TS-QoS scheduling algorithm

Based on the above definitions and methods, the idea of TS-QoS algorithm is as follows: firstly, to compute the priority of tasks based on their normalized attributes in order to reveal the difference of tasks; then, to sort and schedule tasks by their priority; in order to reduce the completion time, in the scheduling process, to set and update MCT matrix by computing the completion time of each task on different services, and search the minimum element value for each task in the row of updating MCT matrix, then, record the special service relating to the minimum element value in the matrix; finally, to schedule the task onto the special service. The

process of TS-QoS algorithm is shown as follows.

```
Algorithm 1: the TS-QoS algorithm
Input: Task(n), Service(m);
Output: the scheduled list.
Begin
   for task set Task(n)
   {   quantify the value of special attributes for task Ti;
       normalize the value of special attributes for task Ti according to the formula (1);
       compute the priority of task Ti according to the formula (2);
   }
   sort task set Task(n) by the priority of tasks;
   initialize ETC matrix and MCT matrix;
   initialize the mapping list;
   for the sorted task set Task(n)
   {   for service set Service(m)
       {   lookup for the minimum MCT(i,j) for task Ti;
           record service Sj;    // service Sj has minimum completion time
       }
       create a new mapping pairing (Ti, Sj) in the mapping list;
       update MCT matrix;
       delete task Ti from task set Task(n);
   }
   for the mapping list
   {   schedule task Ti on service Sj according to the mapping pairing (Ti, Sj);   }
End
```

3. Experimental Evaluation

Cloud computing is an open platform. And it consists of multiple distributed Data Centres. Data Centre constructs all kinds of resource pools by virtualizing physical resources, and encapsulates resources into services. Cloud users submit their requests through a variety of cloud applications established by cloud service providers. Subsequently user requests are formed tasks and are submitted to the Data Centre. And according to the scheduling strategies and the monitored state of resources, Data Centre schedules each task into the appropriate service.

To evaluate the proposed algorithm, we developed an extensive simulation platform based on CloudSim 2.1 to simulate cloud environment by creating Data Centre and many cloud users. We created Virtual Machines (VM) to provide cloud services by Data Centre, and create many cloud tasks based on the requests of users. Then, tasks are scheduled based on the scheduled algorithms. Here, we implement three task scheduling algorithms: the TS-QoS algorithm, the Min-Min algorithm [3] and the Berger model [7]. The machine running CloudSim is configured with Intel Pentium dual-core processor 2.66GHz, 4GB memory.

3.1. The evaluating metrics

In the experiments, we tested the above three algorithm using following common metrics: execution time span (*Makespan*), the average waiting time of long task (*AverageLatency*) and load balancing indexing (δ). The concepts of these metrics are as follows.

(1) *Makespan* is the length of the time interval beginning with the processing of the initial task and ending

with the completion of the last required task. The *Makespan* is a characteristic of the scheduling algorithm. The smaller *Makespan* is, the better the performance of algorithm is.

(2) *AverageLatency* is measured by the ratio of the total waiting time of all long tasks and the number of all long tasks.

(3) δ is a metric that measures whether a system is well load balancing. δ is measured by the standard deviation of load of all nodes. The smaller δ is, the better load balancing is. The formula of δ is as follows:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n}} \tag{4}$$

Where, $n \in N$ is the number of computing nodes, N is a natural number; X_i is the load of the node i, and $\overline{X}$ is the average load of all nodes. In next experiments, a node is a virtual machine.

3.2. Simulation setup

In simulation experiments, we test above metrics for three algorithms with 200~2000 tasks in 50 or 100 virtual machines respectively. The simulated parameters are set as follows.

(1) To set virtual machines according to the description in section 2.1. Where, *processor* is limited in [500, 600]; *bandwidth* is limited in [5, 10]; *storage* is limited in [500, 1000];

(2) To create tasks according to the description in section 2.2. Where, long tasks account for 10% of all tasks. And the size of long tasks is limited in [40000, 60000], the size of common tasks is limited in [10000, 20000]. In all tasks, the A class task account for 10%, the B class task is 20%, the C class task is 70%. And we set the expectation of tasks as follows: *processor* is limited in [400, 800], *bandwidth* is limited in [4, 8], *storage* is limited in [400, 800].

(3) To quantify attributes *TUserType* and *TPriorExp* of task as follows, set *TUserType*$\in$ {1.2, 1.1, 1.0}; *TPriorExp*$\in$ {1.3, 1.2, 1.1, 1.0}.

(4) To set priority weights $\alpha, \beta, \gamma, \omega$ as follows: α =0.4, β =0.3, γ =0.2, ω =0.1.

3.3. Experiments and results analysis

3.3.1 Effect of Makespan

Under the same environment, the tested averages of Makespan of three algorithms for 10 times with 200~2400 tasks in 50 or 100 virtual machines are illustrated in Fig 1.

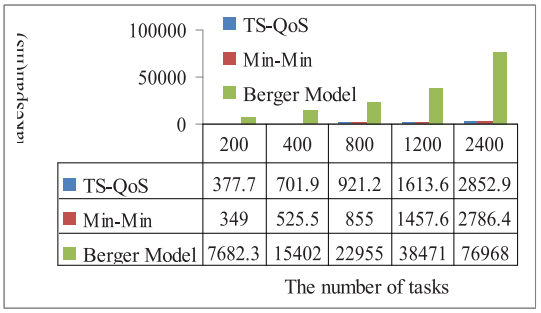
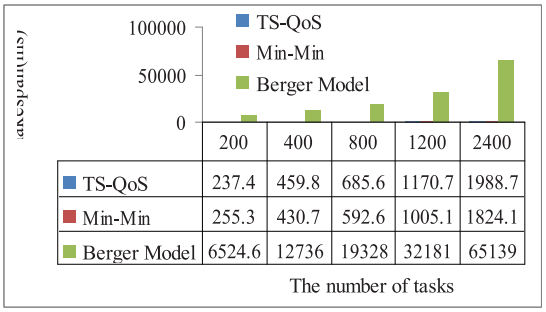


Fig. 1. (a) The tested averages of *Makespan* using 50VM



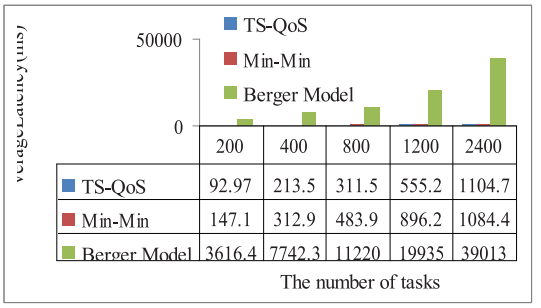
(b) The tested averages of *Makespan* using 100VM

Fig 1 shows that, the *Makespan* in the Berger model is higher than that of the TS-QoS and the Min-Min. It is

because the Berger model always assigns tasks onto the optimal resources in order to satisfy the user requirements for QoS absolutely. This leads that all the tasks were accumulated on few optimal resources. It makes *Makespan* greater. The *Makespan* in the TS-QoS is similar as that of the Min-Min algorithm, because both have same time complexity of $O(n^2 \times m)$ if there are n tasks and m resources. Also, it can be shown that the *Makespan* is reducing with the increase of virtual machines (VM) for the same number of tasks.

3.3.2 Effect of AverageLatency

Under the same environment, the tested averages of *AverageLatency* of three algorithms for 10 times with 200~2400 tasks in 50 or 100 virtual machines are shown in Fig 2.



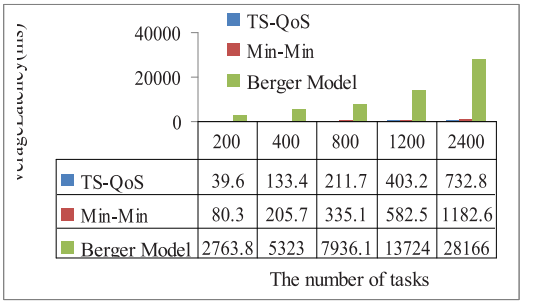
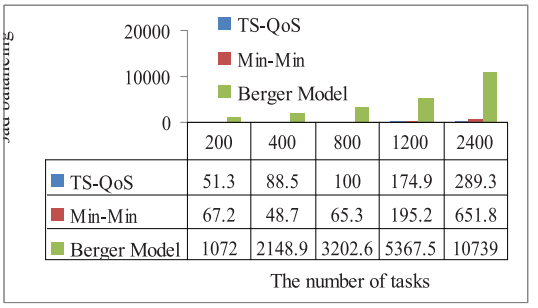


Fig. 2. (a) The tested averages of *AverageLatency* using 50VM (b) The tested averages of *AverageLatency* using 100VM

Fig 2 shows that, the *AverageLatency* in the Berger model is the highest because of its scheduling strategy and lacking of consideration for long tasks. Compared with the Min-Min, the TS-QoS has lower *AverageLatency*. This is because that the length of task has been introduced into the priority, and then long tasks have higher priority and are scheduled earlier. Lower *AverageLatency* helps to reduce *Makespan.*

3.3.3 Effect of load balancing indexing δ

Under the same environment, the tested averages of δ of three algorithms for 10 times with 200~2400 tasks in 50 or 100 virtual machines are illustrated in Fig 3.



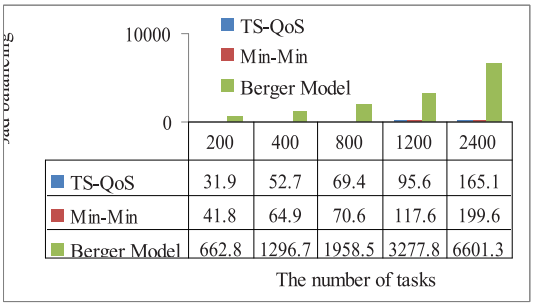


Fig. 3. (a) The tested averages of *load balancing* using 50VM (b) The tested averages of *load balancing* using 100VM

Fig 3 shows that δ in the Berger model is the highest because all the tasks were accumulated on the few optimal resources according to its scheduling strategy. This cause the few optimal resources had the highest load, and the system is load imbalance. And δ in the TS-QoS is lower than that of the Min-Min algorithm. This is because that the TS-QoS algorithm assigns each task onto services which have the minimum completion time in the process of scheduling. In additional, the long tasks have higher priority and are scheduled earlier,

which avoids worsening load balancing. Also, it shows that δ is reducing with the increase of virtual machines (VM) for the same number of tasks.

4. Conclusions

In this paper, we propose a task scheduling algorithm based on QoS-driven in Cloud Computing. The proposed algorithm blends many task attributes including user privilege, expectation, task length and the pending time in queue to compute the priority and sorts tasks by the priority. And the algorithm schedules each task onto a service which has minimum completion time. The experimental results show that the algorithm achieves well performance and load balancing by QoS driving from both priority and completion time.

Acknowledgements

The authors would like to thank the support by the Foundation of Science and Technology on Communication Security Laboratory (9140C110404110C1106), the GuangXi National Natural Science Foundation of China (2012GXNSFAA053224), the Guangxi Graduate Education Innovation project of China (2010105950810M18), and the Foundation of Guangxi Department of Education (201010LX156,CD10066X).

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 2009, 25(6), p. 599-616.

[2] KwangSik Shin, MyongJin Cha, MunSuck Jang, JinHa Jung, WanOh Yoon, SangBang Choi. Task scheduling algorithm using minimized duplications in homogeneous systems. Journal of Parallel and Distributed Computing, 2008, 68(8) , p. 1146-1156.

[3] Braun TD, Siegel HJ, Beck N, etc. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, 2001, 61(6) , p. 810-837.

[4] XiaoShan He, Xianhe Sun and Gregor von Laszewski. QoS guided Min-Min heuristic for grid task scheduling. Journal of Computer Science and Technology, 2003, 18(4) , p. 442-451.

[5] Ching-Hsien Hsu, Zhan, J., Wai-Chi Fang, et al.Towards improving QoS-guided scheduling in grids. 2008 Third ChinaGrid Annual Conference (CHINAGRID). Dunhuang, Gansu, China, 2008, p. 89-95.

[6] Syed Muhammad Ahsan. A framework for QoS computation in web service and technology selection. Computer Standards & Interfaces, 2006, 28(6) , p. 714-720.

[7] Baomin Xu, Chunyan Zhao, Enzhao Hua, Bin Hu. Job scheduling algorithm based on Berger model in cloud environment. Advances in Engineering Software, 2011, 42(7) , p. 419-425.

[8] Xiaoyong Tang, Kenli Li, Renfa Li, Bharadwaj Veeravalli. Reliability-aware scheduling strategy for heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 2010, 70(9) , p. 941-952.

[9] Victor Toporkov, Anna Toporkova, Alexander Bobchenkov, Dmitry Yemelyanov. Resource Selection Algorithms for Economic Scheduling in Distributed Systems. Procedia Computer Science, 2011,4, p. 2267-2276.

[10] Amit Nathani, Sanjay Chaudhary, Gaurav Somani. Policy Based Resource Allocation in IaaS Cloud. Future Generation Computer Systems, 2012, 28(1) , p. 94-103.