Improving Efficiency of HEFT Scheduling Algorithm in Cloud Environment

D. Sumathi[1]
Jayam College of Engg. And Tech.
Dharmapuri

Dr. P. Poongodi[2]
V.S.B. Engineering College
Karur

***Abstract*: In distributed/cloud environment, we need to access large amount of computing, storage and networking resources in virtualized manner. Virtual Infrastructure is a dynamic mapping of system resources to specific applications to maximize utilization. Managing and providing computational resources to large number of users and execution of huge applications is a challenging one for high performance computing system. Before allocating resources, the capacity of application requirements should be calculated and mapped with existing nodes and their load limit. Simultaneously this process needs to consider the overall application cost and good balance between load and availability at each decision point. In cloud computing environment the scheduled target is Virtual Machine resources so the granularity is large and the transferred data is large as well. Equal distribution of hardware resources of virtual machine will improve the running efficiency and the QoS will also meet the client's needs. Scheduling algorithms are used to maximizing resource utilization, minimizing application run time cost, increasing the application availability and fault tolerance by spreading component on allocated nodes.**

Key Words: Cloud Resources, Resource scheduling, HEFT

1.INTRODUCTION:

Cloud computing is known as a provider of dynamic services using very large scalable and virtualized resources over the Internet. Various definitions and interpretations of "clouds" and / or "cloud computing" exist. With particular respect to the various usage scopes the term is employed to, we will try to give a representative (as opposed to complete) set of definitions as recommendation towards future usage in the cloud computing related research space. We try to capture an abstract term in a way that best represents the technological aspects and issues related to it. In its broadest form, we can define a 'cloud' is an elastic execution environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality of service.

There has been various types of scheduling algorithm exist in distributed computing system. Most of them can be applied in the cloud environment with suitable verifications. The main advantage of job scheduling algorithm is to achieve a high performance computing and the best system throughput. Traditional job scheduling algorithms are not able to provide scheduling in the cloud environments

Resource Allocation is all about integrating cloud provider activities for utilizing and allocating scarce resources within the limit of cloud environment so as to meet the needs of the cloud application. It requires the type and amount of resources needed by each application in order to complete a user job. The order and time of allocation of resources are also an input for an optimal resource allocation.

2. CLOUD RESOURCE ALLOCATION:

An important point when allocating resources for incoming requests is how the resources are modeled. There are many levels of abstraction of the services that a cloud can provide for developers, and many parameters that can be optimized during allocation. The modeling and description of the resources should consider at least these requirements in order for the resource allocation works properly.Cloud resources can be seen as any resource (physical or virtual) that developers may request from the Cloud. For example, developers can have network requirements, such as bandwidth and delay, and computational requirements, such as CPU, memory and storage.

When developing a resource allocation system, one should think about how to describe the resources present in the Cloud. The development of a suitable resource model and description is the first challenge that a resource allocation must address. An resource allocation also faces the challenge of representing the applications requirements, called resource offering and treatment. Also, an automatic and dynamic resource allocation must be aware of the current status of the Cloud resources in real time. Thus, mechanisms for resource discovery and monitoring are an essential part of this system. These two mechanisms are also the inputs for optimization algorithms, since it is necessary to know the resources and their status in order to elect those that fulfill all the requirements.

3.CLOUD SCHEDULING ALGORTHM:

3.1. First Come First Server Algorithm

Jobs are served in queue as they come. This algorithm is simple and fast. It is one of the simplest Scheduling algorithms we have it allocate the CPU in the order in which the process arrive. It assumed that ready queue is managed as first in first out which means that the first job will be processed first without other preference

Algorithm FCFS:

- Initialize Tasks.
- First task assigned to the queue and add tasks up to n numbers.
- Add next task 'I' at last position in the main queue.

3.2 Round Robin algorithm

It is one of the oldest, simplest, fairest and most widely used scheduling algorithms, designed especially for timesharing systems [5]. A small unit of time, called time slices or quantum is defined. All run able processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue. The CPU scheduler picks the first process from the queue, sets a timer to interrupt after one quantum, and dispatches the process .If the process is still running at the end of the quantum, the CPU is preempted and the process is added to the tail of the queue. If the process finishes before the end of the quantum, the process itself releases the CPU voluntarily.

Algorithm for RRA:

- Keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time slot, and dispatches the process.
- The process may have a CPU burst of less than 1 time quantum.
- In this case, the process itself will release the CPU voluntarily.
- The scheduler will then proceed to the next process in the ready queue.
- Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the OS.
- A context switch will be executed, and the process will be put at the tail of the ready queue.
- The CPU scheduler will then select the next process in the ready queue.

3.3 Max – Min algorithm

Max-Min is almost same as the min-min al gorithm except the following: in this after finding ou t the completion time, the minimum execution times are found out for each and every task. Then among these minimum times the maximum value is selected which is the maximum time among all the tasks on any resources. Then that task is scheduled on the resource on which it takes the minimum time and the available time of that resource is updated for all the other tasks. The updating is done in the same manner as for the Min-Min. All the tasks are assigned resources by this procedure.

3.4 Min –Min algorithm

Min-Min begins with a set of tasks which are all unassigned. First, it computes minimum completion time for all tasks on all resources. Then among these minimum times the minimum value is selected which is the minimum time among all the tasks on any resources.

Then that task is scheduled on the resource on which it takes the minimum time and the available time of that resource is updated for all the other tasks. It is updated in this manner; suppose a task is assigned to a machine and it takes 20 seconds on the assigned machine, then the execution times of all the other tasks on this assigned machine will be increased by 20 seconds. After this the assigned task is not considered and the same process is repeated until all the tasks are assigned resources.

*Algorithm***:**

1. for all submitted tasks in meta-task; Ti
2. for all resources; Rj
3. Cij = Eij + rj
4. While meta-task is not empty
5. find task Tk consumes maximum completion time.
6. Assign Tk to the resource Rj which gives minimum execution time.
7. remove Tk from meta-tasks set

3.5 Priority scheduling algorithm

This Scheduling algorithm is preemptive in which all things are based on the priority in this scheduling algorithm each process in the system is based on the priority whereas highest priority job can run first whereas lower priority job can be made to wait, the biggest problem of this algorithm is starvation of a process [6].

Algorithm PSA:

```
1 .for i = 0 to i <main queue-size
2. if priority (task i+1 ) > priority (task i )
   then
```

```
3. add task i+1 in front of task i in the queue
4. end if
5. end for
```

3.6 Shortest Job First Scheduling Algorithm

Shortest job First (SJF) also known as Shortest Job Next (SJN) or Shortest Process Next (SPN) is a scheduling technique that selects the job with the smallest execution time [7]. The jobs are queued with the smallest execution time placed first and the job with the longest execution time placed last and given the lowest priority. This Scheduling algorithm is deal with different approach in this algorithm CPU is allocated to the process with least burst time.

Algorithm SJF:

```
1. for i = 0 to i< main queue-size
2. if task i+1 length < task i length then
3. add task i+1 in front of task i in the queue
4. end if
5. if main queue-size = 0 then
6. task i last in the main queue
7. end if
8. end for
```

4. HEFT:

Heterogeneous Earliest Finish Time (HEFT) algorithm for scheduling the tasks of an application, represented by a directed acyclic graph,onto a bounded number of heterogeneous machines[3]. A number of different options for computing the weights in HEFT are considered. In HEFT, a weight is allocated to each node and edge of the graph, based on the average computation and communication, respectively. At that point, the graph is traversed upwards and a rank value is assigned to each node. Tasks are then scheduled, in order of their rank value, on the machine which gives the earliest finish time.

The HEFT Algorithm is an application scheduling algorithm for a bounded number of heterogeneous processors. The algorithm first constructs a priority list of tasks and then locally optimal allocation decisions for each task are made on the basis of the task's estimated finish time. The objective of efficient scheduling is to map the tasks onto the core processors and execution order is set so that task precedence requirements are satisfied and minimum schedule length is given.

The HEFT algorithm is an effective solution for the DAG scheduling problem on heterogeneous systems because of its robust performance, low running time, and the ability to give stable performance over a wide range of graph structures. The limitation of HEFT algorithm is that it uses techniques that are all static approaches of the mapping problem that assume static conditions for a given period of time and also in complex situations it can easily fail to find the optimal scheduling.[2].

.

Bittencount Luiz F. et. al. [8] provides an improvement of Heterogeneous Earliest Finish Time (HEFT) which does not consider the estimates o a single task for the locally optimal decisions but also look ahead in the schedule and consider the information that effect the children of the task by the decisions made. The key idea of this paper is to improve the process of scheduling tasks in HEFT, by looking ahead and considering information about the descendants of a task.

Algorithm:

```
1: compute the average execution time for each task t ϵ Γ
2: compute the average data transfer time between tasks and their successors
3: compute rank value for each task
4: sort the tasks in a scheduling list Q by decreasing order of task rank value
5: while Q is not empty do
6: t ← remove the first task from Q
7: r ← find a resource which can complete t as earliest time
8: schedule t to r.
9: end while
```

The HEFT (heterogeneous earliest finish time) algorithm [9] is a kind of heuristic method based on list scheduling consisting of two phrases: calculating task prioritization and processor selection.

5. IMPROVE EFFICIENCY OF HEFT:

The HEFT algorithm is used to schedule many kinds of tasks which have been put in a workflow, and these tasks have different requirements in terms of resources for successful execution. The goal of HEFT is to minimize the workflow make span but it does not consider factors such as inter-node bandwidth, RAM and storage memory. These factors are included in the proposed algorithm for efficient working of the scheduling algorithm and for better results

HEFT ALGORITHM:

```
1. Compute ranku for all nodes by traversing graph upward, starting from the exit node.
2. Compute rankd for all nodes by traversing graph downward, starting from the start node.
3. jCP j = ranku(ns), where ns is the start node.
4. .For each node ni do
5. If (rankd (ni) + ranku(ni) = jCP j) then
6. ni is a critical path node (CPN).
7. Select the critical-path-processor that minimizes P ni2 CPN wi;j .
8. Initialize the priority-queue with the entry nodes.
9. while there is an unscheduled node in the priority-queue do
10. begin
11. Select the highest priority node from priority-queue,
12. which maximizes ranks (ni) + ranku(ni).
```

```
13. if (ni is a CPN) then
14. schedule ni to critical-path-processor.
15. Else Assign the task ni to the processor pj which
    minimizes the (EFT ) value of ni .
16. Update the priority-queue with the successor(s) of ni
    if they become ready-nodes.
17. end
```

It is also two-phase task scheduling algorithm for a bounded number of heterogeneous processors. The first phase namely, task-prioritizing phase is to assign the priority to all tasks. To assign priority, the upward rank of each task is computed. The upward rank of a task is the critical path of that task, which is the highest sum of communication time and average execution time starting from that task to exit task. Based on upward rank priority will be assigned to each task. The second phase (processor selection phase) is to schedule the tasks onto the processors that give the earliest finish time for the task.

It uses an insertion based policy which considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor, should be at least capable of computation cost of the task to be scheduled and also scheduling on this idle time slot should preserve precedence constraints. The time complexity of HEFT number of tasks in a dense graph and p is the number of processors.

The HEFT algorithm [10] is highly competitive in that it generates a schedule length comparable to the schedule lengths of other scheduling algorithms with a lower time complexity. The HEFT algorithm has two phases: a task prioritizing and a processor selection phase. In the first phase task, priorities are defined as ranku. ranku represents the length of the longest path from task ni to the exit node, including the computational cost of ni , and is given by ranku(ni) = wi+maxnj∈succ(ni){ci,j +ranku(nj)}. For the exit task, ranku(nexit) = wexit. The task list is ordered by decreasing value of ranku. In the processor selection phase, the task on top of the task list is assigned to the processor pj that allows for the EFT (Earliest Finish Time) of task ni .

However, the HEFT algorithm uses an insertion policy that tries to insert a task in at the earliest idle time between two already scheduled tasks on a processor, if the slot is large enough to accommodate the task.

CONCLUSION:

Scheduling is an important and most complicated one in cloud environment. Jobs are scheduled to virtual machine by the job switching center or the cloud data center. Different scheduling algorithms for data center are discussed. The working principle and the efficiency of Heterogeneous Earliest Finish Time Algorithm are analyzed and gave suggestion to improve the efficiency of HEFT by proposing new algorithm Efficient HEFT (EHEFT) which not one consider minimize workflow but also the factors like inter-node bandwidth, RAM and storage memory

REFERENCE:

[1] Renu Bala, Gagandeep Singh (2014),”*An Improved Heft Algorithm Using Multi-Criterian Resource Factors”,* International Journal of Computer Science and Information Technologies, Vol. 5 (6) , 2014, 6958-6963

[2] Chen, W. and Deelman, E. (2012), *“WorkflowSim: A toolkit for simulating scientific workflows in distributed environments”,* 8th International Conference on E-Science, IEEE Publication, 8-12 Oct.2012, Chicago, IL, pp. 1-8.

[3] Zhao, H. and Sakellariou, R. (2008), *“An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm”,* Euro-Par 2003 Parallel Processing , Springer Berlin Heidelberg, Vol. 2790, pp. 189-194.

[4] Rajveer Kaur , Supriya Kinger*,”Analysis of Job Scheduling Algorithms in Cloud Computing”* International Journal of Computer Trends and Technology (IJCTT) – volume 9 number 7 – Mar 2014

[5] Dr Ajay jangra, Tushar Saini*,” Scheduling Optimization in Cloud Computing*,” International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 4, April 2013

[6] Lu Huang, Hai-shan Chen and Ting-ting Hu, "Survey on Resource Allocation Policy and Job Scheduling Algorithms of Cloud Computing" ,Journal of Software, Vol. 8, No. 2, February 2013, pp. 480-487.

[7] Van den Bossche, R., Vanmechelen, K., Broeckhove, J, *“Cost Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads”,* in 3rd IEEE International Conference on Cloud Computing, Miami (July 2010)

[8] Bittencourt, L.F., Sakellariou, R., and Madeira, E.R.M. (2010),*“DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm”*, 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing(PDP), Publication IEEE Conference, 17-19 Feb. 2010, Pisa, pp. 27-34

[9] H. Topcuoglu, S. Hariri, and M. Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, 2002.

[10] H. Topcuoglu, S. Hariri and M. Wu, ”Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, 2002.