DOUBLE LEVEL PRIORITY BASED TASK SCHEDULING WITH ENERGY AWARENESS IN CLOUD COMPUTING

[1] SHACHEE V PARIKH, [2] ASSIS PROF RICHA SINHA

[1]M.E.[IT] Student, Department Of Information Technology, Kalol Institute Of Technology And Research Center, Kalol, Gujarat
[2] Asst.Professor , Department Of Information Technology, Kalol Institute Of Technology And Research Center, Kalol, Gujarat

shachee@ymail.com, richa.872005@gmail.com

ABSTRACT: *Cloud computing is the fastest new paradigm for delivering on demand services over internet and can be described as internet centric software. Scheduling theory for cloud computing is gaining a lot of attention with increasing popularity in this cloud era. Concerns over greening these systems have prompted a call for scheduling policies with energy awareness. The dynamic and heterogeneous nature of resources and tasks in cloud computing is a major hurdle to be overcome for energy efficiency when designing scheduling policies. This paper proposes a priority based scheduling optimization algorithm which addresses these major challenges of task scheduling in cloud. Specifically, our investigation for energy efficiency focuses on: balancing the workload in the way resources are utilized to their fullest and best capacity so that resource power is not left unused. We form a hierarchical scheduler that exploits the multi-core architecture for effective scheduling. Our scheduling approach exploits the diversity of task priority for proper load balancing across heterogeneous processors while observing energy consumption in the system. Simulation experiments prove the efficacy of our approach; and the comparison results indicate our scheduling policy helps improve energy efficiency of the system. This way of resource selection and task selection gives more better results over sequential scheduling.*

Keywords— energy efficiency, dynamic scheduling, task priority, cloud computing .

I: INTRODUCTION

Cloud computing is a very current topic and the term has gained a lot of attention in recent times. It can be defined as on demand pay-as-per-use model in which shared resources, information, software and other devices are provided according to the clients' requirement when needed [1]. Human dependency on cloud is evident from the fact that today's most popular social networking, email, document sharing and online gaming sites are hosted on cloud. Google, Microsoft, IBM, Amazon, Yahoo and Apple among others are very active in this field[5].

Scheduling theory for cloud computing is gaining consideration with day by day hike in cloud popularity. In general, scheduling is the process of mapping tasks to available resources on the basis of tasks' characteristics and requirements. It is an essential aspect in efficacious working of cloud as many task parameters need to be considered for proper scheduling. The available resources should be

utilized efficiently without affecting the service parameters of cloud. Scheduling process in cloud can be generalized into three stages namely–

- Resource discovering and filtering – Datacenter Broker discovers the resources present in the network system and collects status information related to them.
- Resource selection – Target resource is selected based on certain parameters of task and resource. This is deciding stage.
- Task submission -Task is submitted to resource selected[5].

The simplified scheduling steps mentioned above are shown in Figure 1

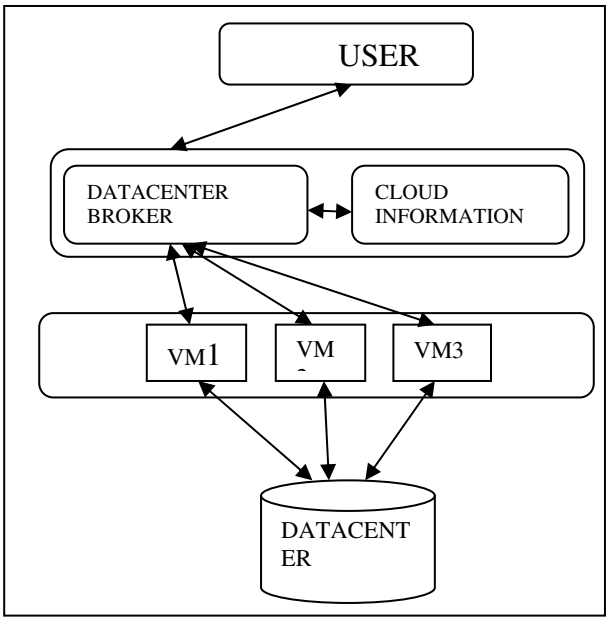

Figure 1 .Scheduling in Cloud

II: RELATED WORK

Target resources in a cloud environment can be selected in various ways. The selection of resources can be either random, round robin, greedy (resource processing power and waiting time based) or by any other means. The selection of jobs to be scheduled can be based on FCFS, SJF, priority based, coarse grained task grouping etc. Scheduling algorithm selects job to be executed and the corresponding resource where the job will be executed.

Generally, resource manager or scheduler can contribute much to the overall energy efficiency of the system. It has the ability to work within the processing requirements/ constraints that put forth by the system users. The processing constraints must be effectively handled particularly in the present of dynamic computing; otherwise it may lead to load imbalance, over-provisioning of resources, and system unreliability. The schedulers in [10, 14] adaptively deal with processing constraints for reliable execution while minimizing energy consumption. The work in [10] developed a software framework to implement and evaluate various scheduling techniques to save energy with the minimal performance impact. The EASY backfilling policy is proposed in [15] to increase resource utilization by continuously monitoring the load in the system. It adaptively selects a certain number of resources that need to be put into 'sleep mode.' In [16] a meta-scheduler is used to select the most energy efficient resource site. The scheduler decides the time slot in which tasks should be executed at the minimum CPU frequency for saving energy. A task consolidation technique is proposed in [9] aiming to maximize resource utilization. The approach contributes for promising energy-saving capability as the energy consumption is significantly reduced when the task is consolidated with one or more tasks. These approaches have demonstrated the effectiveness in minimizing energy consumption while still meeting certain performance goals.

However, the efficacy of these approaches in dealing with system dynamicity is limited to a certain level. Dynamic scheduling can be of a very effective approach for proper load balancing while tracking energy consumption.

III: MODELS

In this section, we describe the application, system and energy models used in our study.

A. Application model

Tasks are assumed to be sequential applications and each of which requires on no more than one core for its execution. The completion time of a task on a particular processor denotes the elapsed time from the time the task arrives into the scheduler until it completes the execution entirely:

$$CT = (wait\_t + exe\_t),$$

where wait_t is the elapsed time between a task submission and the start of execution and exe_t is actual execution time of task, respectively. Tasks arrive in a Poisson process. We assume that the task's profile is available and can be provided by the user using job profiling, analytical models or historical information [12].

Each task ti requires a different processing capacity for its completion; and this determines the priority of that task. We use workload traces from real systems available from the Parallel Workload Archive (PWA) [11] to model the distributed application. From the traces, we obtain submission time, requested time and actual runtime of the tasks. We use the actual runtime as deadline (or upper bound). However, the workload traces do not contain information about processing priority. Hence, we synthetically assign the priority to a given task. The priority is determined based on requested time rt and actual runtime art. A task ti is set to high priority if its art is at least 70% of rt. If art is at most 20% of rt, the priority is considered as low. Otherwise, the task is set to medium priority. Such a task priority considers the consequence of missing deadline[8].

B. System Model

The target system used in this work consists of a number of sites in each of which a set of p heterogeneous processors is fully interconnected (Figure 2). The model allows any set of tasks to arrive at the system and to be executed in available processors. For the sake of simplicity, we form hierarchical scheduler that can handle tasks from

system users and map them onto processors. Hence, there are two types of scheduler: global scheduler GS and local scheduler LS. Specifically, the local scheduler LS of each site is responsible for scheduling tasks after allocation by the global scheduler GS.

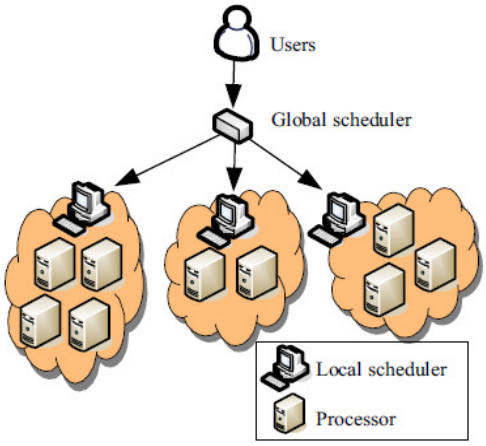

Fig 2. System model

cores. The processing speed of cores in a particular processor is homogenous and expressed in terms of million instructions per second (MIPS). The processing capacity of processor pj is defined as:

$$pc_j = \frac{L}{\sum_{s=1}^{C} cs_s}$$

where L is the total number of tasks completed within some observation period,
cs is speed of core and C is total number of cores in processor pj, respectively.

The processing capacity of a processor is also subject to its availability. If a processor is capable of receiving and executing task considering its energy use the processor is set to available, otherwise it is considered to be unavailable. For a given processor, its processing capacity and availability fluctuate. Therefore, the actual completion time of a task on a particular processor is difficult, if not impossible, to determine a priori.[8]

C. Energy model

The concept of energy efficiency in our approach is defined as the subjective probability by which an processor executes a given task with optimal energy consumption. The efficiency of a processor is subject to task execution time and the utilization. Hence, the energy consumption ECj of a processor pj is defined as

$$EC_j = (p_{\max} * \sum_{i=1} \text{exe\_t}) + (p_{\min} * idle)$$

where pmax is the power usage at 100% utilization (busy), pmin is the power usage when processor j becomes idle and idle is total idle time of pj, respectively. It is assumed that for a given processor the peak power (80 - 95Watt) [13] is proportional to its processing capacity. Typically, the power consumption at idle state is about 50% of the peak power [14]. Hence, we use pmin and pmax of 48 and 95, respectively, which are common for processors used in data centers.

The scheduling scheme requires tracking and adaptive mechanisms to schedule tasks into resources for energy efficiency. The number of incoming tasks in the system is difficult to accurately determine. This dynamic nature contributes to worsening resource performance and increasing processor idle time. In response to this, we introduce the processing power limit for each processor or power-threshold for effective energy use purposes. It helps monitor and discover the level of energy use without significant performance degradation. Different processors have various numbers of cores; and this is a clear indication of varying levels of processing capacity. The powerthreshold ptj is determined differently with two workload scenarios: lightly loaded and heavily loaded.

These scenarios are determined based on the average waiting time in the local scheduler LS during a particular period of time.

The system is set as lightly loaded if the difference between average wait_t and minimum wait_t is at least 80% of average wait_t. Otherwise, it is considered as heavily loaded. Consequently, the power-threshold ptj for lightly loaded denotes the average speed of cores in the processor and for heavily loaded it sets as the total speed of cores, respectively. Since the threshold is related to the number of cores in the processor, load balancing among processors is achieved.[8]

IV: PROPOSED SCHEDULING APPROACH

This section begins by describing our priority-based scheduling strategy and gives details of energy efficiency solution incorporated into our approach.

A. Priority-based scheduling scheme

Schedulers and processors are two main components in the system hierarchy (Figure 3). Queues holding waiting tasks of the global and local schedulers are called globalqueue gq and local-queue lq, respectively. When tasks dispatched from the global scheduler GS reach the processing level, a local scheduler LS assigns these tasks to run in a set of processors. We assume that the local scheduler LS

has all necessary information about processors currently located in the system. Corresponding to two hierarchical schedulers, there are two kinds of scheduling; there are the global scheduling that is executed by GS and local allocation (matching and scheduling) which is performed by LS.

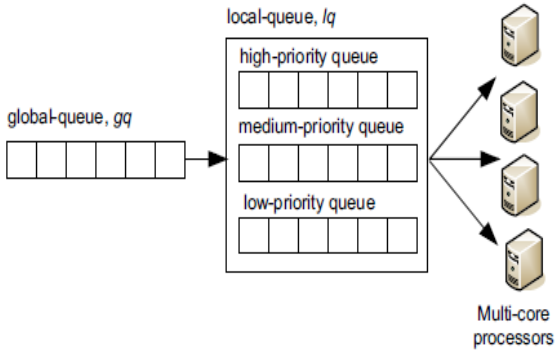

Fig 3. Hierarchical scheduler.

Each and every task submitted first stays in gq. It is assumed that tasks in gq are prioritized by their arrival time (i.e., First Come First Served). For the sake of simplicity, the matching process in the global scheduling is not considered. GS then dispatches tasks to appropriate local schedulers. Local scheduler LS is responsible for matching and assigning those tasks to suitable processors based on the local scheduling policy. The scheduling process at LS is more complicated than that at GS because tasks come with diverse processing priorities. In response to this, we consider three different waiting-queues: low-priority queue (low-pq), medium-priority queue (med-pq) and high-priority queue (high-pq). The local scheduler maintains these queues.

B. Energy consumption constraint

The priority-based scheduling scheme by itself cannot bring efficiency to the energy model. The scheduler needs to effectively schedule tasks in terms of both performance and energy consumption. For each processor, we monitor its energy consumption by controlling the processing requirements that are assigned into it. That means the task assignment is subject to the power-threshold of processor. If a processor reaches its power threshold, the task is assigned into another processor. The power-threshold also exists to control the processor to run at higher energy consumption. Hence, we introduce the processing rate of processor j is given as:

$$PP_j = \frac{\sum exe\_t}{L}$$

LS regularly checks whether processing rate PPj of processor j exceeds its power-threshold ptj. Task assignment is realized when the processing rate of processor is satisfied. The scheduler allows processors to receive and execute other tasks if the energy consumption of processor j is thought to be moderate (i.e., PPj _ ptj). In this state, the processor is considered as available; else, it is marked as unavailable. Obviously, the resource availability fluctuates according to the number of tasks executed by the processors. Due to power-threshold ptj being varied with the load factor (lightly loaded and heavily loaded), the processors able to adapt their processing power in changing the workload. As such, the processing capacity in the system is efficiently used and load balancing is implicitly achieved. The power-threshold of a particular processor contributes to the energy efficiency if PPj on the processor for optimizing resource capacity is actually realized.

Greedy Allocation : Greedy algorithm is suitable for dynamic heterogeneous resource environment connected to the scheduler through homogeneous communication environment [7]. Greedy approach is one of the approach used to solve the job scheduling problem. According to the greedy approach –

Minimum Turnaround Time Based - To improve the completion time of tasks greedy algorithm is used with aim of the resource with minimum turnaround time is given priority than others, resulting in an overall improvement of completion time.

Turnaround Time = Resource Waiting Time + Task Length / Proc. Power of Resource

The task list is rearranged with tasks arranged in descending order of priority in order to execute the task with high priority constraint first. Once the scheduler submits a task to a machine, the resource will remain for some time in processing of that job.

The resource status is updated to find out when the resource will be available to process a new job.

Minimum Cost Based - The resource with minimum cost is selected and tasks are scheduled on it until its capacity is supported. After scheduling each task the resource status is updated accordingly

*Cost of Task = (Task length / Proc Power of Resource) * Resource Cost*

PROPOSED ALGORITHM

The proposed algorithm is optimized which will consider both the constraints deadline and cost.

Double level Priority optimization algorithm.

1. we have major queues as, high , mid and low priority queues.

1st level priority

2.Virtual Machine are sorted in ascending order on the basis of processing power of machine and its cost **.**

2.1 They are divided into 2 or more equal size groups .

3 starting with first group 1,

2nd level priority

3.1 Turnaround time at each resource is calculated Load balancing parameter.

e. If PPj(processing rate) ≤ ptj(power threshold)
processor = available.
else unavailable.

3.2 select the vm with minimum turnaround time and schedule task till the resource capacity is permitted.

4. Update resource status.

V: EXPERIMENTAL DATA

The CloudSim toolkit is used to simulate heterogeneous resource environment and the communication environment [4]. CloudSim(2.1.1) simulator is used to verify the correctness of proposed algorithm. The experiments are performed with Sequential assignment which is default in CloudSim and the proposed algorithm. The jobs arrival is Uniformly Randomly Distributed to get generalized scenario. The configuration of datacenter created is as shown below - Number of processing elements – 1 Number of hosts – 2

Table 1 Configuration of Hosts

RAM(MB)	10240
Processing Power(MIPS)	110000
VM Scheduling	Time Shared

The configuration of Virtual Machines used in this experiment is as shown in Table 2.

Table 2 Configuration of VMs

Virtual Machines	VM 1	VM 2
RAM(MB)	5024	5024
Processing Power(MIPS)	22000	11000
Processing Element(CPU)	1	1

Performance with cost: The tasks execution using the proposed algorithm results in a significant improvement in cost over the sequential allotment as shown in Table 3.

Table 3 Comparison of Execution Cost

No. Of Cloudlets	Proposed Algorithm	Sequential Assignment
25	560.91	735.68
50	1100.82	1471.36
75	1600.73	2207.05
100	2200.6	2942.73

Performance with time: It is evident from the results that proposed algorithm gives better completion time of job in comparison to the sequential approach.

Table 4 Comparison of Task Completion Time

Cloudlets	Proposed Algo	Sequential Algo
25	500.91	735.68
50	1100.82	1471.36
75	1600.73	2207.05
100	2200.6	2942.73
125	900.04	997.99
150	1200.50	1439.75

VI: CONCLUSION

It is observed that the proposed algorithm improves cost and completion time of tasks as compared to Sequential Assignment. The turnaround time and cost of each job is minimized individually to minimize the average turnaround time and cost of all submitted tasks in a time slot respectively. The results improve with the increase in task count.

Energy consumption in LDSs gains a lot of attention recently due to its significant performance, environmental and economic implications. In this paper, we address the energy efficiency issue in the context of scheduling. We have effectively modelled a hierarchical scheduler with the explicitly consideration of urgency of tasks. In addition, the power constraint is devised to effectively control energy consumption of processors with minimal possible performance degradation. Based on our simulation results, our priority-based scheduling demonstrates appealing performance while actively exploiting the heterogeneous nature of both tasks and resources.

The proposed algorithm can be further improved by considering following suggestions –

- Load balancing parameter if needed could be further more improved for proper scheduling of tasks.

REFERENCES

[1] Q. Cao, B. Wei and W. M. Gong, "An optimized algorithm for task scheduling based on activity based costing in cloud computing," In International Conference on eSciences 2009, pp. 1-3. Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[2] Ashutosh Ingole, Sumit Chavan, Utkarsh Pawde. "An optimized algorithm for task scheduling based on activity based costing in cloud computing" (NCICT) 2011, Proceedings published in International Journal of Computer Applications® (IJCA)

[3] Monika Choudhary, Sateesh Kumar Peddoju "A Dynamic Optimization Algorithm for Task Scheduling in Cloud Environment" IJERA ISSN: 2248-9622 Vol. 2, Issue 3, May-Jun 2012, pp.2564-2568.

[4] Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities. By Rajkumar Buyya.

[5] S. Singh and K. Kant, "Greedy grid scheduling algorithm in dynamic job submission environment," in *International Conference on Emerging Trends in Electrical and Computer Technology (ICETECT),* 2011, pp. 933-936.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *Introduction to algorithms*: The MIT press, 2001, pp 16.

[7] Joel H. Ferziger, and Milovan Petric, *"Computational Methods for Fluid Dynamics"*, 3rd Edition, Springer, [2002].

[8] 2011 Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing "Priority-based scheduling for Large-Scale Distribute Systems with Energy Awareness" Masnida Hussin, Young Choon Lee, Albert Y. Zomaya.

[9] Y. C. Lee, and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," Journal of Supercomputing, pp. 1-13, 2010.

[10] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," in Proc. of the 2005 ACM/IEEE Conference on Supercomputing, pp. 34, 2005.

[11] PWA: Parallel workloads archive, http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[12] M. Hussin, Y. C. Lee, and A. Y. Zomaya, "ADREA: A Framework for Adaptive Resource Allocation in Distributed Computing Systems," in 11th Int'l Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Wuhan China, pp. 50-57, 2010.

[13] P. Schreier, "An Energy Crisis in HPC," Scientific Computing World : HPC Projects Dec 2008/Jan 2009.

[14] K. H. Kim, R. Buyya, and J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVSenabled Clusters," in 7th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGRID2007), 2007.

[15] 7B. Lawson, and E. Smirni, "Power-aware Resource Allocation in High-end Systems via Online Simulation," in 19th Int'l Conf. on Supercomputing (Boston, USA), pp. 229-238, 2005.

[16] S. K. Garg, and R. Buyya, "Exploiting Heterogeneity in Grid Computing for Energy-Efficient Resource Allocation," in Proc. of the 17th Int'l Conf. on Advanced Computing and Communications (ADCOM), Bengaluru, India, 2009.