

1. INTRODUCTION

1.1 Cloud Computing

Cloud computing is an Internet-based computing that provides shared computer processing resources and data to computers and other devices on demand. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be rapidly provisioned and released with minimal management effort. Cloud computing is based upon a service-based architecture wherein services are provided mainly at the infrastructure level (e.g., virtual machines, storage) platform level (e.g., database, web server), or software level (e.g., email, ERP solution).

Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in third-party data centers that may be located far from the user—ranging in distance from across a city to across the world. Cloud computing relies on sharing of resources to achieve coherence and economy of scale, similar to a utility (like the electricity grid) over an electricity network.

1.2 Characteristics

The characteristics of cloud computing are:-

1.2.1 Reduction of Cost: There are a number of reasons to attribute Cloud technology with lower costs. The billing model is pay as per usage; the infrastructure is not purchased thus lowering maintenance. Initial expense and recurring expenses are much lower than traditional computing.

1.2.2 Elasticity: Services offered by cloud are rapidly provisioned, and in few cases by itself, rapidly released to quickly scale in. To consumer, the capabilities available for provisioning appears to be endless and could be purchased whenever required.

1.2.3 Security and Availability: The cloud should authorize the data access by the end users. However a fear of security always there with the end users. The requests have to be fulfilled in every case, the data and infrastructure should always available.

1.2.4 Flexibility: Cloud computing mainly stress on deployment of applications in market as quickly possible, by using the most appropriate building blocks necessary for deployment.

1.2.5 Geographical independence: A user can access the data shared on cloud from any location across the globe.

1.3 Issues for Research

To reach the extent of CLOUD computing, major aspects have not yet been developed and realized and in some cases not even researched. Many issues are still needs to be known.

1.3.1 Security of Data: Security of data is a vital and important research issue in cloud computing. It hampers the expansion of the cloud. Usually, the services of cloud computing are delivered by a third party known as service provide, who owns the infrastructure. Even for a virtual private Cloud, the service provider can only specify the security setting remotely, without knowing whether it is fully implemented.

1.3.2 Energy Consumption: To reduce the energy consumption in data centers is another issue in cloud computing. It has been found that that a server even runs a very small workload, Then also it consume over 50% of the peak power. Energy consumption by Google 2,675,898 MWh in 2011 and it is increasing regularly. Thus there is a need to control the consumption of energy otherwise the cost of cloud computing will increase tremendously.

1.3.3 VM Migration: In a cloud environment have more number of data centers, virtual machines have to be migrated between physical machines located in any (same or different) datacenter in order to achieve better provisioning of resources. Migration of VM which means to transfer a virtual machine from one to another physical machines helps in greatly reducing the energy consumption. The benefits for VM migration is it avoid the hotspots.

1.3.4 Data management. As cloud computing is enabling more data-intensive applications at the extreme scale, the demand is increasing for effective data management systems. One main topic in this category is data storage and all the issues that come with it such as data federation (i.e. storage across different providers), data segmentation and recovery, data resiliency, data fragmentation and duplication, and data backup.

1.3.5 Service management: The Cloud as a service-based IT model created a number of challenges pertaining to service management. Service provisioning seems to be at the

core of such challenges. The literature suggests that there is an urgent need for automating service provisioning and making it more dynamic. Another challenge is related to the ability to provide customizable and more context-aware services.

1.4 Motivation

For decades, the high-performance computing community has focused on performance where performance is defined as execution time. To achieve better performance per compute node, microprocessor vendors have not only doubled the number of transistors (and speed) every 18-24 months, but they have also doubled the power densities. Consequently, keeping a large-scale HPC system functioning properly requires continual cooling in a large machine room, thus resulting in substantial operational costs. Furthermore, the increase in power densities has led (in part) to a decrease in system reliability, thus leading to lost productivity.

Typically size of data centers are consists of hundreds or thousands of servers and resources, and the size of data centers are getting a huge expansion due to the rapid increase in use of Cloud computing technology. This rapid growth has made a increase in energy consumed in clouds. With power densities doubling every 18-24 months and large-scale HPC systems continuing to increase in size, the amount of heat generated (and hence, temperature) continues to rise. And as a rule of thumb, Arrhenius' equation as applied to microelectronics notes that for every 10°C increase in temperature, the failure rate of a system doubles.

Thus in order to reduce energy consumption there is a need to effectively utilize the resources and executes the requests. There are various existing methods to manage the workloads but are not able to effectively utilize the resources. The goal of project is to effectively manage the workloads so that the utilization of the resources could be maximized and could redeem the consumption of energy. The less consumption of energy provides a Green Computing environment.

1.5 Task Scheduling

In computing, scheduling is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards

Many parallel applications consist of multiple computational components. While the execution of some of these components or tasks depends on the completion of other tasks, others can be executed at the same time, which increases parallelism of the problem. The task scheduling problem is the problem of assigning the tasks in the system in a manner that will optimize the overall performance of the application, while assuring the correctness of the result.

The task scheduling problem can be modeled as a weighted directed acyclic graph (DAG). A vertex represents a task, and its weight the size of the task computation. An arc represents the communication among two tasks, and its weight represents the communication cost. The directed edge shows the dependency between two tasks.

The primary goal of task scheduling is to schedule tasks on processors and minimize the make span of the schedule, i.e., the completion time of the last task relative to the start time of the first task.

Task Scheduling Aim - A scheduler may aim at one of many goals for Example

- a. Maximizing Throughput – The total amount of work completed per time unit.
- b. Minimizing Response time – Time from work becoming enabled until the first point it begins execution on resources.
- c. Minimizing Latency – The time between work becoming enabled and its subsequent completion.
- d. Maximizing Fairness – Equal CPU time to each process or more generally appropriate times according to the priority and workload of each process.
- e. Minimizing Energy Consumption – Schedule tasks in such a way that they conserve energy more effectively.

To achieve energy conservation concept we propose static DAG scheduling algorithm for heterogeneous environment. Algorithm selects the cluster with the highest upward rank at each step then the cluster is assigned to the most suitable processor that minimizes the energy. Cluster selection phase of the algorithm is based on summation of downward and upward ranks and processor selection of the algorithm is based on the energy consumption and availability of processor.

1.6 Methods of Energy Conservation

There are three ways to conserve energy –

1.6.1 Dynamic voltage-frequency scaling (DVFS) – Dynamic voltage and frequency scaling (DVFS) is the adjustment of power and speed settings on a computing devices various processors, controller chips and peripheral devices to optimize resource allotment for tasks maximize power saving when those resources are not needed. It is perhaps the most appealing method incorporated into many recent processors. Energy saving with this method is based on the fact that the power consumption in CMOS circuits has a direct relationship with frequency and the square of voltage supply. Thus, the execution time and power consumption can be controlled by switching between a processor's frequencies and voltages.

1.6.2 Virtual Machine Migration – In hardware virtualization, physical hardware pieces are carved up into a set of virtual machines — logical hardware pieces that do not have a physical shell or composition, which are essentially just programmed pieces of an overall hardware system. In a virtualization setup, a central hypervisor or another tool allocates resources like CPU and memory to virtual machines. For instance, in older networks, most of the individual elements were physical workstations, such as desktop PCs, which were connected by Ethernet cabling or other physical connections. By contrast, virtual machines do not have a physical interface. They do not have a box or shell or anything to move around. But they can be connected to the same keyboards, monitors and peripherals that humans have always used to interact with personal computers. In virtual machine migration, system administrators move these virtual pieces between physical servers or other hardware pieces. Migration involves moving these virtual machines without shutting down a client system. Modern services often provide migration functionality to make it easier to move virtual machines without doing a lot of other administrative work.

1.6.3 Deactivation– In a cloud system, there are several machines that are not being fully utilized. This underutilization occurs due to variable load on cloud system. These machines which are idle still consume more fifty percent of power consumed by machine

having high load. Thus, in this technique, these underloaded machines are chosen and switched off. The selection phase is an important phase in this technique otherwise results obtained may be unexpected.

2. PROBLEM DEFINITION

We consider power-aware scheduling of bag-of-tasks applications in cloud system. Users submit their jobs and the cloud system should allocate the resource to jobs for the purpose of reducing energy consumption.

The cloud system needs to reduce the energy consumption not only for operational cost but also for system reliability. However, there are some trade-offs between reducing energy consumption and makespan. Running processing elements under low supply voltage decreases the energy consumption but causes jobs to take more execution time due to low processor speeds. On the contrary, controlling processors under high supply voltage can reduce makespan, which incurs much energy consumption. Thus, it is required to control supply voltages of processing elements of the cloud as low as possible to reduce the energy consumption, under the constraint that makespan is reduced. The algorithm deals with the problem of adjusting each processing elements supply voltage as well as scheduling jobs in the cloud system. The proposed approach focuses on the scheduling of currently available jobs in a best-effort manner to reduce makespan and reduce the energy consumption.

As most works on DAG scheduling in the literature, we consider that the programming model and/or the middleware can provide information about the size of each task to the scheduler. These sizes can be obtained by application benchmarks, by a history of executions/input data/data sizes, by estimatives given by the programmer, or by estimatives according to past execution and current data sizes.

3. LITERATURE SURVEY

The method proposed in [1] is two static DAG scheduling algorithms for heterogeneous environments.

- a. Heterogeneous Earliest Finish-Time (HEFT) – It is a heuristic to schedule a set of dependent tasks onto a network of heterogeneous workers taking communication time into account. For inputs HEFT takes a set of tasks, represented as a directed acyclic graph, a set of workers, the times to execute each task on each worker, and the times to communicate the results from each job to each of its children between each pair of workers. It descends from list scheduling algorithms. HEFT executes in two phases – Prioritizing tasks – In the first phase each task is given a priority. The priority of each task is usually designated to be its "upward rank" which is defined recursively as follows $p_i = w_i + \max_{t_j \in \text{children}(t_i)} \{c_{ij} + p_j\}$ where w_i is computational cost of task t_i and c_{ij} is communication cost of between task t_i and t_j .

Assigning tasks to processors – In the second phase tasks are assigned to processors. Now that all tasks are prioritized we consider and schedule each one, starting with the highest priority. The task with the highest priority for which all dependent tasks have finished is scheduled on the processor which will result in the earliest finish time of that task. This finish time depends on the communication time to send all necessary inputs to the processor, the computation time of the task on the processor, and the time when that processor becomes available.

- b. The CPOP Algorithm – The Critical-Path-on-a-Processor (CPPOP) Algorithm is another heuristic for scheduling tasks on a bounded number of heterogeneous processors. The rank_i and rank_j attributes of nodes are computed using mean computation and communication costs. The CPOP Algorithm uses $\text{rank}_i(n_i) + \text{rank}_j(n_i)$ to assign the node priority. The processor selection phase has two options: If the current node is on the critical path, it is assigned to the critical path processor (CPP); otherwise, it is assigned to the processor that minimizes the execution completion time. The latter option is insertion-based (as in the HEFT algorithm). At each iteration we maintain a priority queue to contain all free nodes and select the node that maximizes $\text{rank}_i(n_i) + \text{rank}_j(n_i)$. A binary heap was used to implement the priority queue, which has time complexity of $O(\log v)$ for insertion and deletion of a node and $O(1)$ for retrieving the node with the

highest priority (the root node of the heap). The time complexity of the algorithm is $O(v^2 * p)$ for v nodes and p processors.

The method proposed in [2] is a combination of VM placement algorithm to reduce energy consumption and modified Max-min algorithm (which aims to reduce make-span). VM placement is done in two phases: the first phase deals with the admission of new requests for VM provisioning and switching off the nodes or hosts which do not meet the VM resource requirements (like CPU, memory, Hard Disk etc.). The remaining physical nodes are ranked based on their resource availability. Higher rank is assigned to the node which has greater resource availability. The number of nodes switched off increases linearly with number of VMs when CPU, memory, Hard Disk requirements are considered and is more than when only CPU requirements are considered. In the second phase all the VMs are sorted in increasing order based on the power requirements. The amount of power consumed by the resources is related to the capacity used. Therefore, this ensures that the VM from the sorted list which provides least increase in power consumption is allocated to the node with higher rank. The proposed algorithm saves about 24% of the energy consumed using existing energy aware algorithm. The limitation is that when VM's are placed on physical machines then there is no way describe that tells the priority of one VM with other.

In [3], the author describes Path Clustering Heuristic which is a DAG scheduling heuristic that uses the clustering technique to generate groups (clusters) of tasks and the list scheduling technique to select tasks and processors. The PCH groups paths of the DAG and schedules them initially on the same resource, with the objective of reducing communication costs. This scheduling heuristic has shown good performance when communication between tasks (communication to computation ratio—CCR) is medium or high. The first step of the PCH algorithm is to compute, for each task, some attributes based on information given by the middleware and by the DAG specification. With the initial values of the attributes calculated, the algorithm starts to create clusters of tasks to begin the scheduling process. It uses the priority attribute to select the first task to be added to the first cluster. The first node t_i selected to compose a cluster is the unscheduled node with the highest priority. It is added to cluster, then the algorithm starts a depth-first search on the DAG starting on t_i , selecting successors of t_i with the highest rank and adding it to the same cluster, until it has a non scheduled predecessor.

In reference [4] a method is proposed that deals with scheduling problem in cluster systems to minimize the QoS degradation in terms of meeting deadlines. Processing Elements (PEs) are assumed to be homogeneous so that they provide the same processing performance in terms of MIPS. When a cluster system receives a job from a user, the resource controller decides whether to accept the job. The proposed job admission scheme guarantees the deadlines of previously accepted jobs in the system. The steps include:

- a. Job submission: A user submits a new bag-of-tasks job with deadline to the cluster system.
- b. Schedulability test : The resource controller requests schedulability for each task of the job to all PEs.
- c. Acknowledgement of schedulability : Each PE tests the schedulability of the new task and returns the estimated energy consumption in case of being schedulable.

Selection of PEs: The resource controller selects the lowest-energy PE which can run each task.

In reference [5] a method is proposed to reduce energy consumption in parallel task. This method reduce power consumption in 2 phases-

- a. Method to reduce power consumption in processing phase-For each PE, it scans all time slots. When the PE is idle or transfers data in a time slot, scales the PE's operating frequency to the lowest. When a time slot executes a non-critical job, it calculates its slack time, extends the job's execution time to the slack time, and scales down the PE's operating frequency to a proper value. This algorithm can achieve up to 35% energy saving in simulation.
- b. Method to reduce power consumption in communication phase –It proposed Task clustering algorithm that reduces the make-span by zeroing edges of high communication costs. It firstly marks all edges as unexamined and allocate each task a separate cluster. After sorting all edges in descending order of communication time, the PATC algorithm repeatedly merges tasks by zeroing the edges with high communication cost if the total power consumption is not increased. This algorithm can achieve up to 30% energy saving in simulation.

The method proposed in reference [6] for energy optimization in heterogeneous embedded multiprocessor systems. It divides the method in 2 parts.

- a. Task scheduling and voltage scaling (TSVS),
- b. Task Mapping (TM)

In this a schedule is first generated based on an initial TM. In each optimization step, try to remap a task to a new processor and/or voltage level such that reduce the total energy consumption of the schedule as much as possible while decreasing the slack/increasing the make-span as little as possible. In this way, we are optimizing the TM as well as the TSVS at the same time based on the current partially optimized schedule. In doing so, optimized energy-efficient schedule can be get in a shorter time. This algorithm was able to reduce average energy consumption by 9%-10%.

4. PROPOSED METHODOLOGY

4.1 Main Idea

Considering the literature survey and the limitations of the existing algorithms we propose static DAG scheduling algorithm for heterogeneous environment. Our main objectives are

1. Energy conservation
2. Reduce make-span

Algorithm consists two phases

- a. Task Scheduling** – The task scheduling problem can be modeled as a weighted directed acyclic graph (DAG). A vertex represents a task, and its weight the size of the task computation. An arc represents the communication among two tasks, and its weight represents the communication cost. The directed edge shows the dependency between two tasks. To schedule parallel tasks we make clusters of tasks according to their priority and task dependencies and assign these clusters to processor which gives minimum finish time. We schedule clusters on processors to achieve minimum make-span.
- b. Energy Conservation-** The power reduction can be done with the help of the technology called DVFS. The system would automatically scale down the frequency and supply voltage of the CPU in order to reduce power consumption, as power consumption

is proportional to the CPU frequency and to the square of the CPU supply voltage. After Task scheduling and voltage scaling it is checked if the deadline of all the tasks are met. If Yes, this schedule is fixed otherwise another task schedule is generated with possibly lower energy consumption.

Definition of the attributes

Attribute	Definition
EST	The earliest time is the time the activity can begin
EFT	The earliest time is the time the activity can end
ET	The execution time is the time spent by the system executing the task
W_i	The computational time of the task
C_{ij}	The communication cost between tasks
P_i	The priority of the task
$N=\{t_1, t_2, \dots, t_n\}$	No of the tasks
$PR=\{pr_1, pr_2, \dots, pr_p\}$	No of the processors
$C= \{c_1, \dots, c_c\}$	Clusters of the tasks
$R = \{r_1, r_2, \dots, r_r\}$	No of the resources

4.2 Methodology

The first step of the algorithm is to compute, for each task, some attributes based on information given by the middleware and by the DAG specification. As most works on DAG scheduling in the literature, we consider that the programming model and/or the middleware can provide information about the size of each task to the scheduler. These sizes can be obtained by application benchmarks, by a history of executions/input data/data sizes, by estimatives given by the programmer, or by estimatives according to past execution and current data sizes. Initially we assign the priority of the tasks and select the task with highest priority and make clusters. then we put that successor of the task whose all parents are clustered and highest priority are put in same clusters in which cluster their parents resides.

4.2.1 Priority Calculation

The task attributes used by the algorithm are defined as follows:

Earlier Start time (EST):- EST is the time the activity can begin. In other word we can say the earliest point in time when the schedule activity can begin (based on preceding logic and constraints). Alternative names: Early Start Date (ESD) or Early Start (ES).

In our algorithm the EST can be calculated based on availability of the resource. If it is starting task then by allocating that task on the resource then EST of that task is the minimum time at which the resource is available. Otherwise the start time of the task will be the time at which all its predecessor tasks has been scheduled and after that the minimum time at which resource is available.

if task is starting task **then**

$$EST(T_i, r_k) \leftarrow Time(r_k)$$

else

$$EST(t_i, r_k) \leftarrow Time(r_k) + EST_{pred}$$

Here EST_{pred} is the maximum of $EST_k + W_k + C_{k,i}$ of every predecessor of task t_i and $Time(r_k)$ is the earliest time when resource k is ready for task execution.

Earlier Finish time (EFT):- EFT is the time the activity can end. In other word we can say the earliest point in time when the schedule activity can complete (based on preceding logic and constraints) Alternative name: Early Finish Date (EFD) or Early Finish (EF). In our algorithm the EFT can be calculated based on EST of the task and ET (Execution Time) of the task on a processor. EFT is calculated by adding task execution time with task start time.

$$EFT(t_i) \leftarrow EST(t_i) + ET(t_i)$$

Computation cost (W):- Computation cost of the task is the ratio of number of instructions in the task and power consumed by the resource. Calculate the computation cost of every task according to power consumed by available

resources. $W_{i,r} = \text{instruction}_i / \text{power}_j$. Here instruction_i is the number of instructions in the task and power_j is the power consumed by resources.

Communication Cost (C):- Communication cost is the time for passing the message between tasks. Communication cost is the cost between task t_i and t_j using link between resources and task. Communication cost between task t_i and task t_j is calculated by $C_{i,j} = \text{data}_{i,j} / \text{bandwidth}_{r,t}$.

Priority of Tasks (P):- Priority decides the importance of the task. If two tasks that are scheduled for the same time and the same resource, then we need to rank them to decide which one to execute first. This 'might' be due to time (ie: one affects other downstream work), or it could be by duration (one will take a day while the other a week), or it could be deadline (one HAS to be done by 12/24). In our algorithm we assign priority to the tasks based on computation time and communication time of the tasks.

Execution Time (ET):- The execution time or CPU time of a given task is defined as the time spent by the system executing that task. In our algorithm the execution time represent the estimate execution time of the task. It is defined by – $ET(t_i, r_j) \leftarrow (W(t_i) * CPI) / f_{j,k}$ where task t_i executing on resource r_j at some frequency $f_{j,k}$ and CPI is the number of cycles per instruction of r_j which is based on on architecture of the computer and instruction set.

4.2.2 Clustering

With the initial values of the attributes calculated, the algorithm starts to create clusters of tasks to begin the scheduling process. The algorithm uses the priority attribute to select the first task to be added to the first cluster (clustering phase). The first node (or task) t_i selected to compose a cluster cl_k is the unscheduled node with the highest priority. It is added to cl_k , then the algorithm starts a depth-first search on the DAG starting on t_i , selecting $t_s \in \text{succ}(t_i)$ with the highest P_s and adding it to cl_k , until t_s has a non scheduled predecessor. The task t_s that has a non scheduled predecessor is not included in the cl_k . With this, clusters always have

only tasks with all predecessors already scheduled or to be scheduled along with them. For each cluster created, the algorithm selects a resource to schedule it. After creating clusters the algorithm must decide where it will be scheduled. In the processor selection phase, the algorithm looks for the resource which minimizes the EFT of the cluster. Thus, the criterion to choose the processor to a cluster cl_k is to minimize the EFT_{cl_k} , defined as $EFT_{cl_k} = \text{maximum of } EFT_{t_i}$ where $t_i \in cl_k$

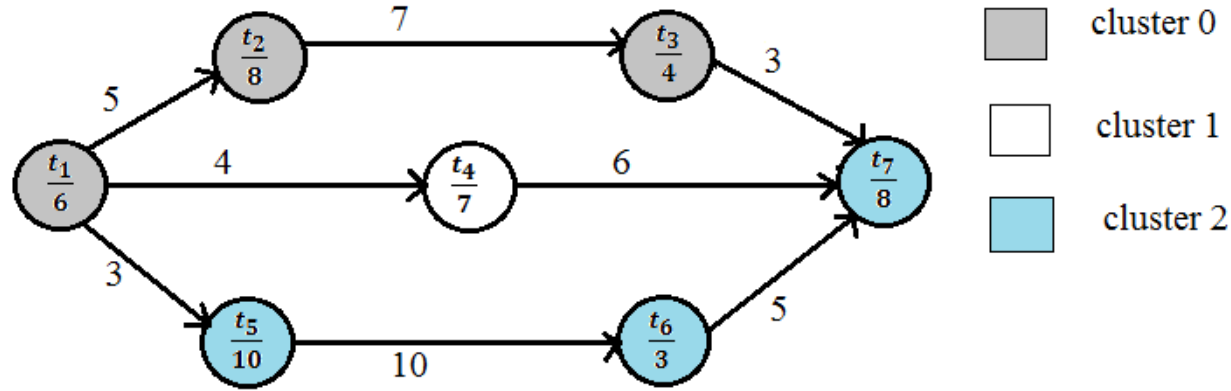


Fig.4.2.2.1 Clustering of Tasks in DAG

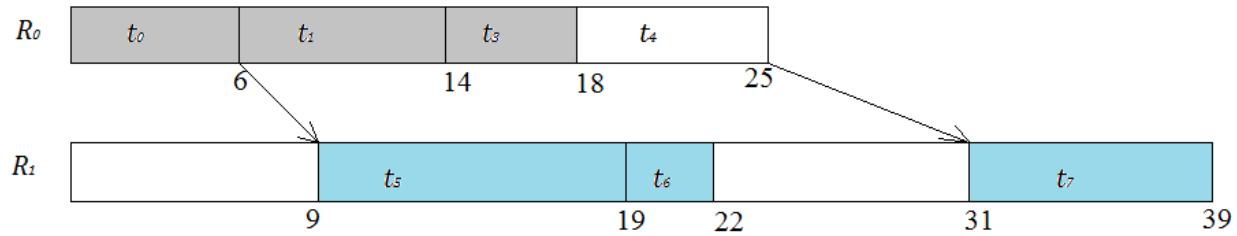


Fig.4.2.2.2 Scheduling of Tasks on Processor

4.2.3 Voltage Scaling

After assigning a processor to every cluster. We need to find total energy consumption. So that by using DVFS total energy consumption can be minimized. So for every task in cluster find execution time and find total execution time of that cluster by adding execution time of each task in that

cluster. Idle time of each cluster can be calculated by subtracting total execution time of each cluster from time at which its corresponding processor works. For Execution time and idle time processor runs on maximum and minimum frequency respectively. So overall energy can be calculated by-

Overall Energy Consumption = coefficient of processor * (max voltage)² * execution time + coefficient of processor * (min voltage)² * idle time. Now the energy can be optimized using DVFS. In DVFS the execution time of a task on a processor gets increased, by lowering the frequency of the processor for that particular task. For that we find set the frequency of processor so that task finish time gets nearest less than or equal to the start time of next task on that processor. Execution Time = Weight of Task * Cycles Per Instruction of V_m / Frequency of that V_m. Overall new Energy Consumption = for every task i on that processor (coefficient of processor * (DVFS voltage_i)² * execution time_i) + coefficient of processor * (min voltage)² * idle time

Level	Voltage	Relative Speed
0	1.5	100
1	1.4	90
2	1.3	80
3	1.2	70
4	1.1	60
5	1.0	50

Fig 4.2.3.1 Voltage levels and respective relative speed of processors

Inputs –

1. MIPS & CPI of every processor
2. MI of every Task
3. No of Tasks
4. Bandwidth of communication link

5. Task dependency between Tasks

4.3 DVFS ENABLED TASK SCHEDULING ALGORITHM

1. For task i from 1 to n do
 2. calculate priority p_i by calling priority() function
 3. do clustering of tasks by calling clustering() function
 4. Schedule cluster of tasks on processor which gives minimum EFT by calling shedulingcluster() function
 5. for processor k from 1 to p
 6. do dvfs(p_k) by calling dvfs() function

priority() function

1. for i from 1 to n do
2. if t_i is the last task
3. then $p_i \leftarrow w_i$
4. else
5. $p_i \leftarrow w_i + \max_{t_j \in succ(t_i)} (c_{i,j} + p_j)$
6. end for

Clustering() function

1. sort task acc. to their priority
2. For j from 1 to n
3. Label:
4. Add task t_j to Cluster cl_k
5. Sort children of task t_j
6. for i from 1 to $child_i$ do
7. if t_i is not in cluster and has no non scheduled predecessor
8. Then
9. $T_i \leftarrow t_j$
10. Goto label

11. end for
12. If($j=p+1$)
13. $K++$;
14. end if
15. end for

Scheduling cluster() function

1. for every cluster cl i 1 to c do
2. $EFT_i \leftarrow INT_MAX$, $chosen_vm$
3. for Every vm j to p
4. $EFT_{i,j} \leftarrow INT_MIN$
5. for every task k of cluster i
6. Calculate EFT of $T_{k,i}$ on vm j
7. If $EFT(T_{k,i}) > EFT_{i,j}$
8. $EFT_{i,j} \leftarrow EFT(T_{k,i})$
9. endfor
10. if $EFT_{i,j} < EFT_i$
11. then $EFT_i \leftarrow EFT_{i,j}$
12. $chosen_vm \leftarrow j$
13. endif
14. endfor
15. endfor

dvfs function()

1. for i from 1 to ct do
2. $newET_i \leftarrow oldET_i + EST_{i+1} - EFT_i$
3. $fp \leftarrow oldET_i * max_fp / newET_i$
4. round off fp to nearest frequency level

5. Hardware & Software used

5.1 NetBeans IDE 8.2 with jdk 8:- NetBeans IDE supports JDK 8 features, such as lambda expressions, repeatable annotations, compact profiles, etc. When these constructs are used in our code, the IDE recognizes them, correctly highlights errors, and lets we automatically fix syntax. Thus, NetBeans IDE helps us write code that is compatible with Java SE 8 Release Contents Early Draft Review Specification. After JDK 8 is downloaded and installed on your system, it needs to be registered in the IDE as follows:

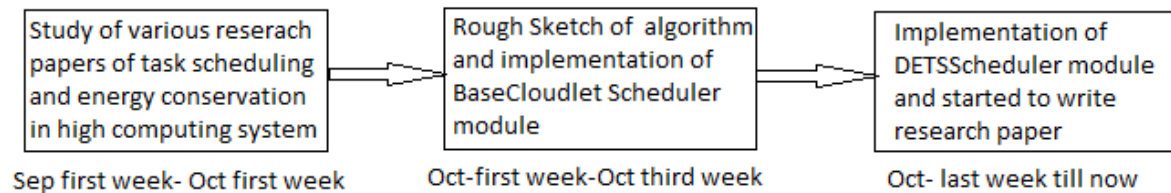
1. In the IDE, choose Tools > Java Platforms from the main menu.
2. Click Add Platform in the Java Platform Manager dialog.
3. In the Add Java Platform dialog, select Java Standard Edition and click Next.
4. Specify the directory that contains the JDK and click Next.
5. Verify that the default locations of the Platform Sources zip file and API documentation are valid. Click Finish to close the Add Java Platform dialog box.
JDK 8 is registered as a platform in the IDE
6. Ensure JDK 1.8 is chosen in the Platforms list and click Close.

5.2 Cloud Sim:- CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. Originally built primarily at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, The University of Melbourne, Australia, CloudSim has become one of the most popular open source cloud simulators in the research and academia. CloudSim is completely written in Java.

6. Activity Time Chart

Till mid semester evaluation, we had studied about various task scheduling algorithms that have designed till now. We also studied various techniques that are can be used to reduce energy consumption in high computing system. We had presented the sketch of our algorithm. We had initiated BaseCloudletScheduler module which contains various functions that a general scheduling algorithm contains.

After mid semester evaluation, we had developed DEFTScheduler module and Clustering module. DEFTScheduler class extends BaseCloudletScheduler class. We had developed various functions required in this class. The functions of Clustering module have also been developed. We have also started writing a research paper for the same.



7. Results and Discussion

7.1 Assumptions:

- Limited size of dataset
- Information regarding tasks and machines provided
- Processors can operate at 6 voltage levels

7.2 System Description:

The DETS algorithm constitutes of the following functionalities:

- Calculate_Priority
- Clustering
- Voltage Scaling
- Energy_Calculation

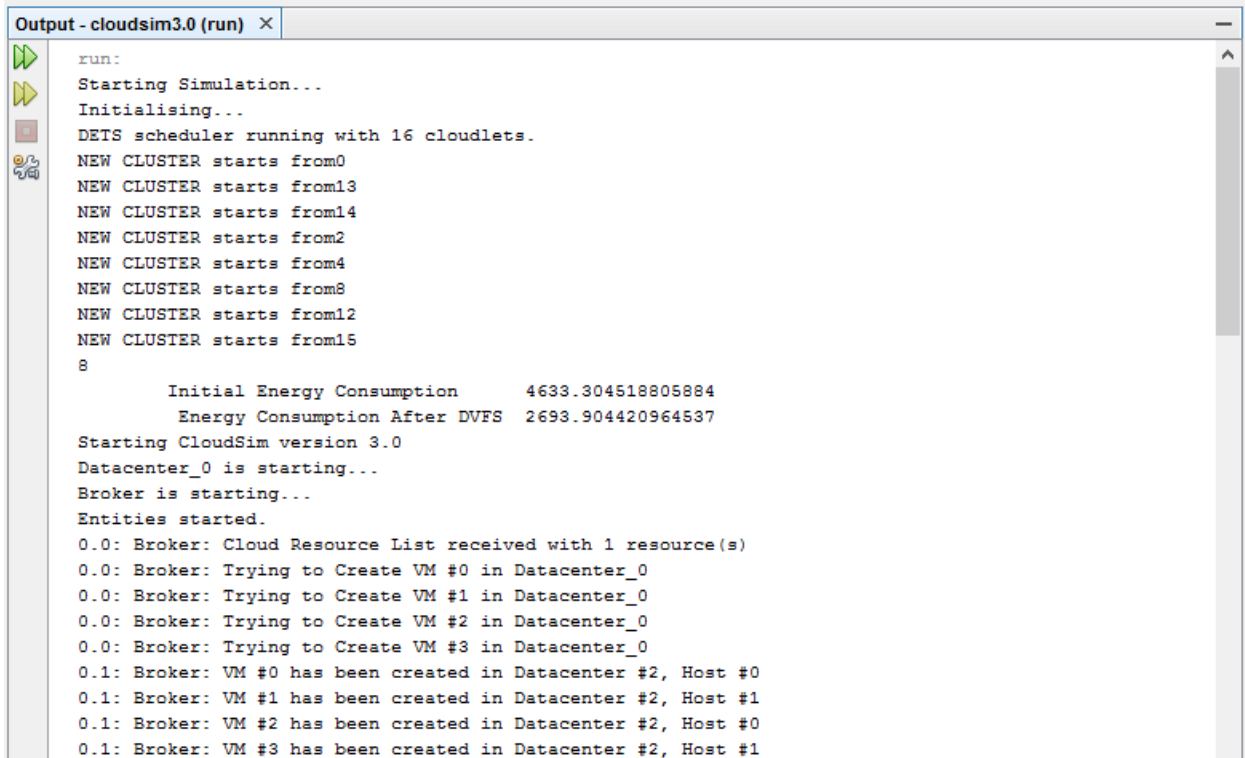
7.3 Analysis

- This algorithm has been tested on two set of tasks. One set of tasks contains 8 tasks which are to be scheduled on 2 processors. 40% energy reduction has been observed this case. Another set of tasks consists of 16 tasks that need to be scheduled on 4 processors. In this case, the energy reduction has been found to be 41.25%.Complexity Analysis
Complexity of algorithm = $(n + c) + n + (p * n) + n \sim (p * n)$

n=number of tasks

p=number of processors

c=number of communicational link



```
Output - cloudsim3.0 (run) x
run:
Starting Simulation...
Initialising...
DETS scheduler running with 16 cloudlets.
NEW CLUSTER starts from0
NEW CLUSTER starts from13
NEW CLUSTER starts from14
NEW CLUSTER starts from2
NEW CLUSTER starts from4
NEW CLUSTER starts from8
NEW CLUSTER starts from12
NEW CLUSTER starts from15
8
Initial Energy Consumption      4633.304518805884
Energy Consumption After DVFS   2693.904420964537
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.0: Broker: Trying to Create VM #3 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #1
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: VM #3 has been created in Datacenter #2, Host #1
```


8. Future Scope of our work

This algorithm assumes only one incoming workflow in the cloud system. It can be extended for multiple workflows where there are various entry and exit tasks instead of only one. Moreover the present algorithm considers only medium computational to communicational cost ratio. Changes can be adopted to consider variable computational to communicational cost ratio.

REFERENCES

- [1] Z. & S. W. Yu, "An adaptive rescheduling strategy for grid workflow applications," *IEEE International Parallel and Distributed Processing Symposium*, vol. 3, no. 4, pp. 1-8, 2007.
- [2] M. & B. Dhanalakshmi, "Task Scheduling Techniques for minimizing Energy Consumption on Response Time in Cloud Computing," *International General of Engineering Research and Technology*, vol. 3, no. 7, pp. 547-557, 2014.
- [3] L. F. & M. E. R. Bittencourt, "Towards the scheduling of multiple workflows on computational grids," *Journal of grid computing*, vol. 3, no. 8, pp. 419-441, 2010.
- [4] K. H. B. R. & K. J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters," *CCGRID*, vol. 7, pp. 541-548, 2007.
- [5] L. T. J. v. L. G. & C. D. Wang, "Power Aware scheduling for parallel tasks via task clustering," *Parallel and Distributed Systems (ICPADS)*, pp. 629-634, 2010.
- [6] L. K. V. B. & V. S. Goh, "Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 20, pp. 1-12, 2009.

