JOB SHARING STRATEGY FOR PRIORITIZED TASKS IN MOBILE CLOUD COMPUTING ENVIRONMENT

ABDUL KHADEER, BIKSHPATHI DADI, J RANJITH
Assistant Professor, Department of Computer Engineering, Ellenki college of Engineering and Technonlogy, patelguda (vi), near BHEL ameenpur (m), Sangareddy Dist. Telangana 502319
.

ABSTRACT
In recent times, users necessitate and expect more demanding criteria to perform computational intensive applications on their mobile devices. Based on the mobile device limitations such as processing power and battery life, Mobile Cloud Computing (MCC) is turned to be a more attractive choice to influence these drawbacks as a mobile computation can be provided to the cloud, which is coined as Mobile computation offloading. Prevailing researches on mobile computation offloading determines offloading mobile computation to single cloud. Moreover, in real time environment, computation service can be offered by multiple clouds for every computation services. Therefore, a novel and an interesting research crisis in mobile computation offloading begins with, how to choose a computation service for every tasks of mobile computation like computation time, energy consumption and cost of using these computation services. This is also termed as multi-site computation offloading in mobile cloud computation. In this investigation, offloading computation to diverse cloudlets/data centres with respect to task scheduling is formulated for examination. So, a heuristic algorithm known as Accelerated Cuckoo Search Algorithm based job sharing is designed to attain higher data transmission rate in the MCC. The results of the anticipated method outperform the prevailing methods in terms of effectual job sharing; transmission speed, Bandwidth used, execution time of a job, transmission value, through put value, buffering overhead and reduced waiting time. The simulation was carried out in Cloudsim environment for superior output.

Keywords: Computation Offloading, Mobile Cloud Computing, Scheduling, Heuristic Algorithm, Job Sharing.

1. INTRODUCTION

With the rising popularity of individual mobile devices such as smart phones and tablet PC and with booming wireless Internet access, huge amount of newer mobile applications related to complex computation like high quality video and 3- D interactive games are also emerging. These sorts of applications generally need higher computational capacities like memory and computation hardware (task scheduling and resource allocation). To assist mobile users requirement over these application types, powerful computational capacities are likely associated with mobile devices.

Nonetheless, in conventional mobile computation systems, storage capabilities and computation over mobile devices are often restricted. For certain resource-related applications and energy-intensive applications, hosting them over mobile devices is a challenge. Mobile cloud computing is known as a promising task [1]-[2] to address this crisis, in which users can offload the computation tasks to external cloudlets from their mobile devices via internet to improve resource insufficiency of mobile devices [3]. As utilizing the cloud service, mobile users can also allocate their jobs with diverse requirements on services and offer payments to cloud services for service usage through pay-per-use methodology.

Mobile users in general, has to pay to attain better Quality of Service from cloud provider with their financial budgets, to provide better services for users, cloud providers has to provision huge resources for hosting job and therefore more costs is incurred. In specific, there is resource competitions amongst the users' jobs concurrently submitted to cloudlets. In certain, mobile cloud computing environment, the cloud provider spotlight on, how to schedule these simultaneous jobs request effectively [4] over physical resources in cloudlets to enhance its profit in a trivial manner.

In this investigation, initially the problem for maximizing the job sharing amongst the cloudlets has been explored. For instance, consider a parallelizable task scheduling computation jobs provided to multiple cloudlets [5]. One foremost feature of this type of jobs is that the execution can be quickened up to an extent by facilitating huge resources to them. Indeed, for this types of job sharing, waiting time and response time are the most significant quality of service criteria for evaluating how services can be delivered and how mobile users can pay more for certain job to be finished earlier and vice versa. While a cloudlet encounters burst resource demands, there may be certain jobs submitted cannot be assigned with any resources. Therefore, the jobs will be in a queue till they are provided with a resource. But still, the burst can remains for higher duration that causes long waiting time in queue before served by the available cloudlets. The profit of these kinds of jobs is reduced to a greater extent rapidly, as there is a delayed service response time incurred. Sometimes guaranteed deadline for job execution may be violated. Based on these circumstances, effectual job sharing approaches are needed for the cloud providers to allocate these jobs simultaneously submitted by numerous mobile users.

So as to resolve the above crisis of job sharing, task scheduling and reduced waiting time in mobile cloud computing, this work anticipates an efficient cuckoo search algorithm for job sharing approach in mobile cloud. Initially, evaluate the task to be scheduled to the cloudlets [21] and their available resources based on the priority of the job. Subsequently, passed on the priority condition, waiting time of jobs will be changed with the elapsing waiting time with respect to make span value [22]-[23]. Finally, the efficiency and the effectiveness of the anticipated work have been tested via simulation and experimentations. A significant contribution of this investigation is to attain ideal computation time for scheduling the task based on priority by unlimited waiting in certain degree.

Contribution of the Work:

The main contributions of this investigation are as follows:

1. The performance and the effectual Job sharing strategies over multiple cloudlets, multi-core based computation offloading problem is formalized initially, associated with NP-hard evaluation.
2. A novel heuristic algorithm [ACS-JS] is designed to enhance task scheduling and job sharing while satisfying the time constraint of numerous applications.
3. The anticipated scheme was evaluated based on the offloading algorithm with simulation and discussion on the MCC. The outcomes show that the proposed method can offer extremely effectual job sharing with reduced waiting time for resources.

Organization of the Work:

The remainder of the work is ordered as trails: section II explains the state of the art associated with the mobile cloud computing task scheduling strategy and the scheduling approaches. Section III gives a detailed explanation of the proposed methodology. Section IV presents the detailed results and discussion associated with the anticipated method. Section V gives the conclusion and the future work extension, followed by the references.

2. RELATED WORKS

S.Zhang et al., [6] anticipated dynamic-need based planning approach for enhancing cloud supplier's benefit in mobile cloud computing situations. The idea of equilibrium point from financial hypothesis is utilized to register the initial priority of job, and after that, revise the waiting jobs priorities based our proposed time subordinate dynamic-priority method. The author conducted examinations and simulation dependent on synthetic datasets and the outcomes demonstrate that the proposed dynamic- priority based scheduling method can acquire preferable execution over the static priority scheduling methodologies. It recommends that the proposed model is powerful in enhancing cloud suppliers' benefit in mobile cloud computing conditions.

X.Lin et al., [7] describes MCC task scheduling issue. To our best learning, this is the main scheduling work that limits energy consumption under hard completion time constraint for task chart in MCC condition, considering the joint task scheduling on neighbourhood centres in mobile phone, remote communication channels, and cloud. A novel calculation is recommended that begins from a negligible delay scheduling and hence performs energy reduction by migrating tasks among neighbourhood centres and cloud. A linear time rescheduling calculation is proposed for task migration with the end goal that the general computational complexity is adequately diminished. Recreation results exhibit energy reduction with the general completion time requirement fulfilled.

Y.Geng et al., [8] distinguished new research difficulties in computation offloading presented by the big LITTLE design. Rather than just choosing locally or remotely running task in the conventional design, the author considers how to abuse new engineering to limit energy while fulfilling application completion time requirements. The author tended to issue of energy effective calculation offloading on multicore-based mobile phones running different applications. Y.Geng et al., [8] previously formalized the multicore-based computation offloading issue, and after that proposed heuristic based solution for together offloading choice and task scheduling issues. To locate superior task schedule to fulfil both application time constraint and task-dependency prerequisites, while diminishing schedule searching overhead, Y.Geng et al., [8] proposed critical path based solution which recursively checks tasks and moves tasks to correct CPU centres to spare energy. Y.Geng et al., [8] assessed the proposed computation through simulation and real applications on Android. The trial and simulation results exhibit that the calculation can altogether diminish energy utilization of mobile phones while fulfilling application time limitations.

C. Tseng et al., [9] proposed portal based edge computing service replica and GAF task scheduling component that enables edge gateway to plan more tasks inside short average waiting time. The resource estimation and lightweight VNF design advances are utilized to enhance operational efficiency of edge computing services, to build resource usage, to accomplish a quick organization of VNF, and to meet service on-demand requests. The mix of resource estimation, task scheduling and lightweight VNF setup configuration gives coordinated arrangement that can fulfil service on- demand requests for 5G systems. The simulation results demonstrated that proposed GAF component performs subsequent scheduling algorithms. Meanwhile, examination uncovered that utilizing lightweight virtualization innovation in edge gateways is more effective and focused than utilizing conventional VMs.

L.Liu et al., [10] anticipates novel task scheduling model and novel TSFC calculation of fog computing condition are proposed dependent on I- Apriori calculation. Association rules are created by I-Apriori calculation which goes about as vital parameter of TSFC task scheduling algorithm. Test results demonstrate that TSFC calculation has preferred execution over other similar algorithms as far as total execution time execution time and average waiting time.

M. Joshi et al., [11] explained about the framework is an authoritative level framework which implies it has usage range. The range can be reached out to more elevated amounts like urban communities or states. M. Joshi et al., [11] can expand the list of application or tasks

to be offloaded every once in a while according to necessities. This usefulness has
been created just for android mobile phones, it very well may be additionally produced for iOS based devices like iPhones, iPad and so forth and furthermore Windows based mobile phones. Additional scheduling criterion can be connected for much enhanced outcomes.

L.Tianze et al., [12] dissected task scheduling problem for ad-hoc based mobile edge computing and anticipated an overhead-optimizing task scheduling component. The task scheduling component accepted open opportunistic utilization, energy utilization, time delay, and money related expense into record, targets for limiting overhead for mobile phones. Li exhibited communication model and computing model, at that point examined overhead in both offloading task and local computing examples. Next, task assignment issue as a multi-device task scheduling and demonstrated that game was potential diversion. At that point, an overhead-optimizing task scheduling calculation was formulated, and the computational complexity was analyzed. Finally, simulations were led to assess energy of proposed plan, and execution was contrasted and other three schemes. The outcomes demonstrated that anticipated task scheduling plan could adequately limit overhead of mobile phones and effectively entire tasks.

O.Chabbouh et al., [13] advancement of an entire offloading procedure for H-CRAN made of offloading decision and task scheduling for request to enhance network recital and client QoE. In this manner, O.Chabbouh et al., [13] handle offloading choice and offloading request scheduling for Cloud-RRH. To begin with, O.Chabbouh et al., [13] proposed dynamic multi-parameter offloading decision plan with end goal to adjust offloading choice to present system state and application characteristics. At that point, scheduling mechanisms was created as linear programming optimization capacity that plans to diminish total execution cost communicated as over-burden, network delay, and relocation.

H.Qian et al., [14] exhibited Jade, a framework that empowers computation offloading from mobile phones to cloud. Jade can successfully decrease energy utilization of mobile phones, and powerfully

change its offloading procedure as indicated by device status. H.Qian et al., [14] assessed Jade with two applications: a face location application and path finding application. Results demonstrated that Jade can adequately diminish energy utilization for two applications while enhancing their execution.

S.Banerjee et al., [15], ends up that the proposed algorithm diminishes makespan of hosts to great extent thus algorithm can be actualized progressively to give a superior QoS. The proposed calculation can additionally be reached out for allotting VMs with different intelligent information and thus makespan can be diminished further with assistance of legitimate smart programmed virtual resource management. Something else should be focussed is the movement of VMs. The closest administration area to client is critical to look after QoS. Finally, security concern about cloud data is likewise one of the essential goals.

3. PROPOSED METHOD

System Model

In the system model, application is specified using a DAG (Directed Acyclic Graph) G = (V, E). Every node in the network is provided as Vi ε V signifies task and directed edge ‘e’, where (Vi, Vj) ε E specifies precedence constraint, which represents that node (task) Vi should finish its execution before node(task) Vj commences its execution. There is some amount of ‘N’ nodes (tasks) in task graph. Assume that, in a task graph, task without any parent is termed as entry task and task [7] without any children is termed as exit task. As depicted in fig 1, task V1 is entry task whereas task V10 is exit task. For every task Vi, define datai, which is the sum of task specification and data input needed to be uploaded to cloud [24] and the sum of data to be downloaded from cloud, if task execution, Vi is offloaded onto cloud.

Figure 1: Graphical Representation of Task Precedence

Mathematical Formulation

This section describes a mathematical model for load balancing and allocation problem based on Cuckoo Search Algorithm. Objective of this formulation is to form a load aware allocation

$U_j = Load_{ij} \le Load \le Load_{ij}$ (4)

Multi-core utilization (cluster of nodes) as in equation 5, 6 and 7:

C_{ij} =[task length/resource processing Power] * Resource utilization cost (5)

Where,

u 1/β

strategy. The objective function here is to allocate the task to the virtual machine so as to achieve minimum execution time, minimum cost and meet the deadline constraints with a well-balanced load across all processors. Let Eij represents the amount of time taken by Task 'i' to execute in resource 'j' and Cij denotes the computational cost. This

Task lengthj = 0.01 * (j)
vj

Where,
U = ϕ. randomn [d]; V = randomn [d];

* V- Xbest (6)

problem can be expressed as linear programming problem, as depicted below in equation 1,

$E_{ij} = C_{ij}$ (1)

computation in resource 'j' and Yij specifies the computational cost. This crisis may be expressed as linear allocation of jobs as depicted below in equation 2 and 3:

randomn [d] ☐ produc
between [0,1]

Then,

V = V + task length * randomn [d] (7)

Once the task is completed by a cloudlet, updat the next proces to perform computation as given below 8:

Xbest ☐best f (X f (Xi) (8)

Thus, the mathematical model provides the way to assign task based on the priority by frequent update.

Priority Based Job Sharing for Effectual load Balancing in Cloud VM

In this section, a dynamic priority based job sharing

L (X) = Max ∑n

n j=1

P (Xij)

(2)

procedures has been generated to deal the jobs submitted with the objective of reducing the

In this section, a mathematical model is described for balancing the load to the available cloudlet for effectual job sharing. The foremost objective of this section is to form a Load aware Job Sharing strategy. The objective function is to assign task to virtual machine (VM), to achieve minimum complexity, minimum execution time and to fulfil the deadline constraint with well-balanced load and effective job sharing to all processors. Let Xij specifies the total time utilized by task 'i' to carry out

complexity of computation. As given in the previous section, each job submitted to the cloudlets is related to the complexity function described by mobile users. Based on the response time for allocating job when the current job terminates, the job sharing criteria for submitted job and the value associated with it provides the probability of computing the complexity. Priority is based on both the probability of scheduling time slot unit and probability of virtual resource unit criteria.

When waiting queue is not empty, the priority can be revised based on the job as proposed in the Priority based Job sharing strategy. With the subsequent scheduling time slot, waiting jobs in the queue with new priority will compete with the resources with new arriving jobs, thereby waiting jobs priorities will be updated. Thus, the dynamic priority based job sharing is modelled based on following measures:

So as to address the crisis of Job sharing and task scheduling more clearly, partition the scheduling time interval based on priority in sequence of hours provided by t = 0,1,2 and so on.

The resources capacity in the MCC environment is always re-calculated during the end of each scheduling interval. Based on the priority of every job submitted into cloud is computed before the commencement of the next scheduling slot.

Scheduling merely occurs during the initial phase of scheduling slot. Scheduling cannot be carried out during an interval of execution.

The priority based scheduling strategy presented in this work is non-primitive, once the resources are provided to execute a job shared in the cloud environment, it cannot be pre-empted by subsequent jobs till the job terminates its execution.

In general, transmission time for cloud to transmit the outcome of computation performed to mobile user can be eliminated as the volume of data of computational outcome is generally much lesser than input data volume.

Prioritizing Task Based On Job Sharing Strategy

In this section, the priority to every task is computed and explained. Initially, calculate the computation cost CCi for every task. If task Vi is cloud task, its computation cost CCi is provided as in equation 9,

CCi = T re (9)

Where,

T □ prioritized task;

If task Vi does not belongs to cloud task, CCi is computed as average computation time of task in local cores as in equation 10, i.e.

CCi = average Tl (10)

Then, priority level of every task is defined recursively by equation 11,

Priority (vi) = CCi + maximumvj ε successive(vi) priority (vj) (11)

Where,

Successive (vi) □ task successors;

Priority levels are computed recursively using task graph traversing that commences from exit tasks. For every task exit, priority level is equal to as in equation 12,

Priority (vi) = CCi for Vi ε exit tasks (12)

Followed by this section, this work illustrates the proposed heuristics algorithm, Accelerated Cuckoo Search for Job sharing to share job to available resources in cloud environment.

Accelerated Cuckoo Search Algorithm for Job Sharing

ASC-JS algorithm is sourced from the absolute brood parasitic behaviour of cuckoo species with the aggregation of Levy flight behaviour of the bird as in figure 2. Let this section explains the interesting breeding characteristics of certain accelerated cuckoo species. Thereby, the work outlines the significant concept and idea of ACS-JS algorithm.

Accelerated Cuckoo Breeding Behaviour

Cuckoos are attractive birds; however their aggressive reproduction approach is extremely interesting to us. Cuckoos lay their eggs in shared nests which are selected using levy flight. They habitually select nest where host bird merely laid its own eggs. To enhance probability of hatching the own eggs, they possibly remove other eggs. Certain female cuckoos imitate patterns and colours of eggs of host species. Cuckoos diminish probability of eggs being discarded to increase their re-productivity. Therefore, when host bird discovers weird egg, either it will throw them or leave nests and move somewhere to produce new one. Cuckoo often selects nest with eggs in it.

The subsequent section explains cuckoos levy flight.

Algorithm

```
Objective function f(x), x=( x1, …, xd);
Initial population of 'n' host nests xi (I =1,2,…,n); while (t < Maximum Generation);
Get Accelerated cuckoo (i.e. I ) randomly and produce novel solution using Lévy flights;
Compute its quality/fitness; Fi
Choose nest among n (i.e. j ) randomly; if (Fi > Fj),
Replace j by new solution; end
Discard fraction (Pa) of worse nests
[produce new ones at new locations through Lévy flights];
Keep best solutions (or nests with finest solutions); Rank solutions and obtain current best;
End while
Post process results
```

Here, in this algorithm 'Z' is vector with entries Zi (t+1) specifying next cuckoo generation from Zi, where ith one is evaluated by equation 13,

$$Z(t+1) = Z(t) + \alpha \oplus \text{Levy}(u) \qquad (13)$$

$\alpha > 0$
the problem given;

$\oplus$□ Entry-wise multiplications

The successive steps of cuckoo basically produce random walk process which provides power-law
i.e. task-length distribution based on its heavy tail. In real world scenario, if cuckoo's egg is extremely related to host's eggs, thereby cuckoo's egg is slightly less to be identified. As a result, it is effectual idea to carry out random walk in appropriate way with certain random task length.

Levy Flights

The general behaviour of animals to hunt for food is a quasi-random way. Currently, various investigations have provided flight behaviour of numerous insects and animals that demonstrating typical features of Levy flights. In general, animals foraging path is successfully a random-walk as subsequent move is sourced on both present transition probability and location/state to next location. Selecting direction is implicitly based on probability which can be designed accurately. Diverse studies have illustrated that flight behaviour of numerous insects and animals illustrates typical features of Levy flights. Next, certain behaviour has been functional for optimization and optimal search; preliminary outcomes offer promising capability.

Pseudo-code

The primary steps of cuckoo algorithm cast off in this investigation comprises of:

1. Identify Cuckoo [25] present location haphazardly
2. Allocate to every cuckoo eggs
3. Identify each cuckoo eggs' radius
4. Cuckoos lay eggs in hosts nests that are in range [26]
5. Eggs detected by host birds are destroyed

i i 6. Seeds that have not been recognized are
□ task length, based on the scalability of
Levy (u) = t^λ, $1< \lambda<3$ Where,

bred
7. Estimate Cuckoo's novel location

8. Accelerated Cuckoo method with finest groups Cuckoo recognizes purpose as residence place

9. New population Cuckoo travels to current location
10. If you stop condition established stop, otherwise go to subsequent step
11. Cuckoo in simulation procedures
12. At this stage of benchmark examination, Accelerated Cuckoo algorithm delivers task scheduler.
13. Make span minimization work is the significant goal
14. Input: List of Cloudlet (tasks) and list of VM
15. Output: Finest solution for allocation of job to VM

Job Sharing Aware Cuckoo Based Allocation

The issue of finding task of least make span and cost is NP-hard. Because of job sharing complexity issue, most investigators anticipated meta-heuristic procedures for resolving the crisis. Here, Accelerated Cuckoo Search for Job sharing procedure is utilized to locate an optimal outcome. Objective function designed is consolidated is in co-operated with fitness function where it will then be utilized to quantify the execution with regards to objective algorithm. The fitness function can be computed by equation 14,

$$P(X_{ij}) = aU_j - bY_{ij} - cX_{ij} \qquad (14)$$

The detailed functionality of the proposed algorithm is depicted as trails. At first, node aggregated for computation is done over cloudlets. Physical resources (PMs) inside cloudlet are enclosed as racks and are regularly composed as group of thousands of hosts for allocating resources purposes. Here, for nodes density based computation is performed, where physical machine capacity is taken as a measure for aggregation. In this investigation, it is expected that every single virtual clusters and virtual machines resides in same cloudlet, despite the fact that they may be on various racks.

It is considered that transmissions in networks between nodes are constant. As job-sharing is carried out by Accelerated Cuckoo based method, foremost thing is to initiate population of probable solutions. Jobs appear at unknown intervals are placed in waiting queue of tasks scheduled from where jobs are allocated to processor. Each time, when job arrives at scheduled tasks queue (task pool), jobs are scheduled using effectual scheduling algorithm and placed in appropriate queue. Before allocating, load on every cloudlet has to be analysed, and nodes should be classified as slightly loaded, Overloaded, and Under Loaded VM. If under loaded, VM tasks are consolidated to slightly loaded VM and freed VM are permitted to move to sleep mode for reducing energy consumption [16] [20]. Therefore, queue of slightly loaded VM is maintained in a cloudlet.

The job sharing strategy initially verifies the information of slightly loaded VM available in the waiting queues. During job arrival, dispatcher evaluates waiting queue. If the queue is non-empty, dispatcher eliminates slightly loaded processor first from queue and assists job to lightly loaded VM. If queue is empty, it is identified whether entire resources can host VM instance. If it is probable, new VM instance is generated and job is deployed to specific VM. If both conditions are not fulfilled, job is assigned to randomly selected processor by Accelerated cuckoo search. After job assignment, load balancing is carried out. If node is seems to be overloaded, two scenarios are evaluated. Intra PM and Inter PM. In Intra PM, overloaded VM instance is measured and load is moved to slightly loaded VM within PM. In Inter PM, task is moved to PM whose average utilization is smaller than threshold within similar node. By evaluating, architecture of most cloudlets, default CPU utilization of 100% is measured as threshold.

Algorithm

Step 1: Number of tasks/node ( ) Step 2: Job sharing ( )
Step 3: At job arrival to VM, dispatcher consults waiting queue for priority.
Step 4: If (waiting queue) ≠ ø

```
Dispatcher removes first VM from queue and allocates job to this specific VM.
Step 5: If (waiting queue) = ø
{
If (global Resources Can Host VM), then
{Initiate new VM instance Add VM to available VM list Deploy services to new VM
}
Else {
Dispatcher directs job to randomly chosen nodes using Accelerated Cuckoo ( )
Job sharing ( )
}
Step 6: If (VM available) { Intra Pm ()
Find overloaded VM instance and transfer load into least loaded VM within Pm.
}
Else { Inter Pm ()
Migrate task to Pm whose average utilization is lesser than threshold within same node.}
```

4. RESULTS AND DISCUSSION

So as to examine the effectiveness and efficiency of the anticipated Accelerated Cuckoo Search algorithm for Job sharing, extensive experiments and simulations were performed in Cloudsim environment. Simulation outcome was utilized as it facilitates to perform iterative experimentation, and measuring computational complexity than real time execution. Besides, the proposed approach with dynamic priorities based scheduling approaches in virtual resource unit. The experimental outcomes cast off that anticipated model perform better than prevailing methods. This section illustrates experimentation through numerical simulations to evaluate efficiency and effectiveness of proposed research.

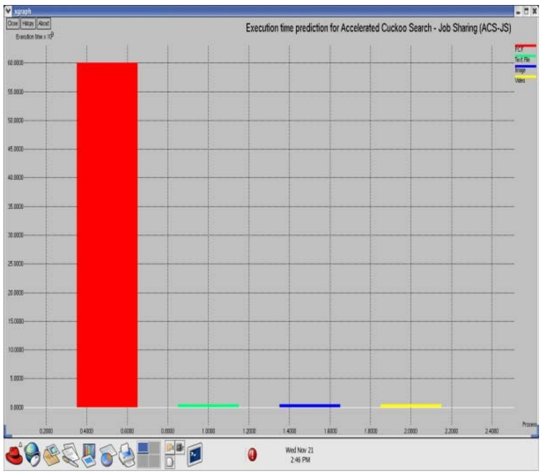
Experiment Setup

The experiment was performed on the laptop with 8192M of RAM, 2.5GHz Intel CPU, Microsoft Window7 OS. The algorithms are executed in CloudSim environment and computation carried out in diverse parameter settings, like energy utilization [17]-[18], task scheduling and load balancing [19]

in ACS-JS. Task graph structure was initialized owing to random simulation. For each task, the workload is generated randomly and data transmitted into subsequent tasks.

Figure 3: Graphical Representation of Task Execution Time of Proposed ACS-JS

Fig. 3 and fig.4 shows the execution time of various tasks offloading strategies in MCC environment. Task such as PDF, text file, Images and video are offloaded. While the time computation for the process to be carried out in

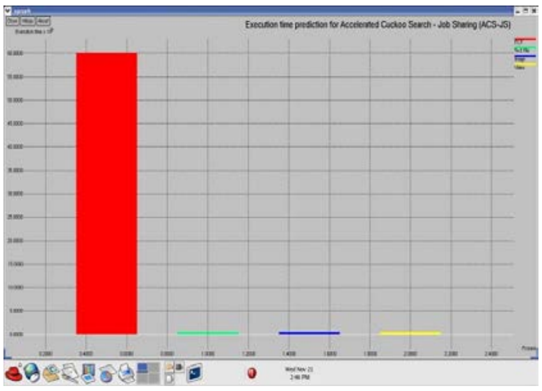


MCC is shown in the above figure.

Figure 4: Graphical Representation of Task Execution Time of EBCO-TS Vs ACS-JS

As discussed in previous phase, first step of ACS- JA is initializing population. This is carried out using vector, where vector length is measured as sum of resources. The choice was made as solutions
produced are less than 'k' generations may not be appropriate. Subsequently, executing ACS-JS for more than 'k' generations may not extremely feasible, as too much leads to computational overhead. Eij signifies amount of time taken by taski to perform in resource Rj and Cij specifies computational cost. After initializing initial population haphazardly, finest nest is selected.

Next, Levy flight function is carried out and fitness function is utilized to attain finest nest. The experimental outcomes of proposed approach are shown below. Test runs were grounded on default value, except for number of tasks which are facilitated to vary. Higher Threshold multiplier is fixed as 1.2 and Lower Threshold multiplier is fixed as 0.8



The algorithm is compared with existing algorithm by evaluating the factors parameters like percentage of deadline constrain, CPU utilization and computational cost makespan. With respect to the comparison and outcomes from experiment, it is concluded that anticipated approach works better. The investigation of result recommends that Accelerated cuckoo search based job sharing strategy outperforms well for reducing makespan and computational cost and percentage of deadline constrain strategy is higher in contrast to other Meta heuristic algorithm.

Figure 5: Throughput Computation of Proposed ACS-JS Vs EBCO-TS

Fig. 5 depicts the throughput value of the proposed ACS-JS algorithm with the existing EBCO-TS heuristic approach. The proposed method throughput value= 22.

Figure 6: Bandwidth Value Computation of Proposed ACS-JS Vs EBCO-TS

ID	CLOUDLET	MAKE SPAN	FLOW TIME
1	0	5	2
2	2	3	2
3	1	0	6.5
4	3	0	1

Fig. 6 depicts the bandwidth consumed for allocating task by the proposed ACS-JS algorithm with the EBCO-TS heuristic approach.

Figure 7: Buffering Overhead (Transmission Overhead) Computation of Proposed ACS-JS Vs EBCO-TS

Fig.7 depicts the transmission overhead for allocating task by the proposed ACS-JS algorithm with the EBCO-TS heuristic approach. Theoverhead of the proposed method is lower than the existing method.

Figure 8: Transmission Value Computation of Proposed ACS-JS Vs EBCO-TS

Fig. 8 depicts the transmission value for allocating task by the proposed ACS-JS algorithm with the existing EBCO-TS heuristic approach. The transmission rate of the proposed method is higher than the existing method.

Figure 9: Graphical Representation of Task Scheduling using ACS-JS

Fig.9 depicts the task scheduling strategy for allocating job by the proposed ACS-JS algorithmwith the existing EBCO-TS heuristic approach. The scheduling strategy of the proposed method is higher than the existing method.

Fig.10 depicts the recovery time for next job by the proposed ACS-JS algorithm with the existing EBCO-TS heuristic approach. The priority of the proposed method for task allocation is higher than the existing method.

CPU Utilization

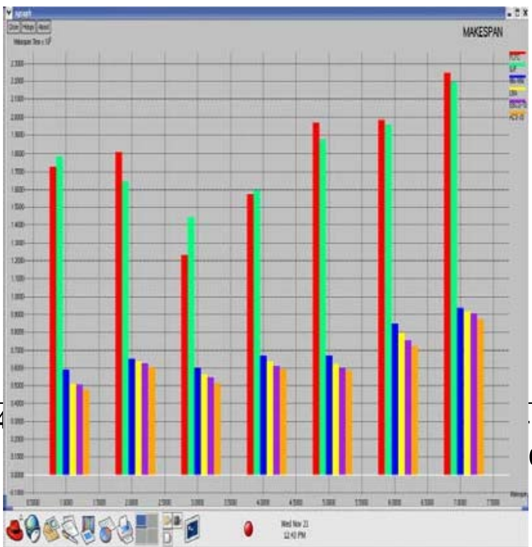
In the anticipated approach, new VMs are recognized and incoming tasks are assisted to new virtual machine if the VM is overloaded, while running the allocated jobs. Using this proposed approach, diverse virtual machines are uniformly loaded at certain points and idle VMs are utilized efficiently. Fig. 12 shows the graphical representation of utilizing the CPU resource for performing various tasks.

Makespan Evaluation

Makespan is distinct as total execution time of entire tasks in nodes as in equation 15.

Makespan = $\sum n$ $ExecutionTime$ ($Task$i) (15)

Makespan is needed to have lower value. Existing algorithm fails to attain the target as it consumes huge time to perform execution. By



comparing with existing algorithm, it is obvious that makespan of Accelerated cuckoo search algorithm is lesser in contrast to prevailing algorithms. The execution times for diverse amount of tasks are recorded for cuckoo search. Fig.13 shows the makespan time utilization of the proposed work with the existing algorithms such as FCFS, SJF, Min-Min, LBA, EBCO-TS with the proposed work. The anticipated ACS-JS shows better trade off than the prevailing algorithms.

Figure 13: Graphical Representation of Makespan Utilization using ACS-JS

Minimal Priority Rejection

To examine the algorithm based on meeting user required deadline, percentage of deadline constraint for every algorithm has to be plotted. Entire comparison recommends that Accelerated Cuckoo Search Based Job Sharing strategy is foremost appealing algorithm for scheduling as presented in this work. Percentage of deadline constraint by Accelerated cuckoo search is considerably higherthan prevailing techniques. Tardiness TR evaluates percentage of deadline constraint as in equation 16:

$$TR = di - fi \quad (16)$$

Total Tardiness is sum of tardiness of every task which did not performed under provided deadline. The sum of non-delayed tasks is total amount of tasks whose completion time is lesser than task deadline, i.e., which completed within given deadline. The assumed completion time is measured as mean of completion time for task at each resource as in fig.14.

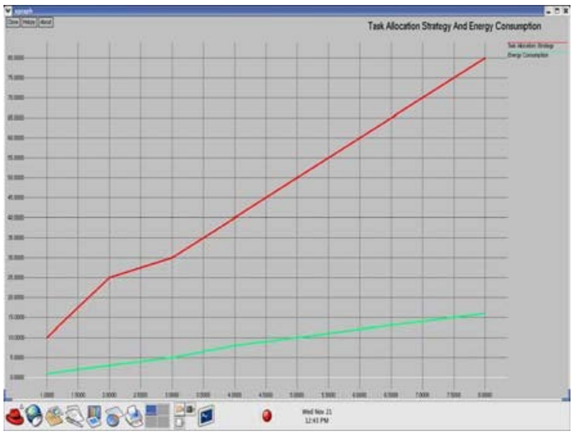


Figure 14: Graphical Representation of Task Scheduling using ACS-JS

5. CONCLUSION

In this investigation, an effectual Accelerated Cuckoo Search Based Job Sharing Algorithm is presented. The scenario is modelled as a heuristic optimization problem which attempts to reduce overall job sharing and computational cost, meeting deadline constraints, maintaining load in VM has to be balanced. The problem was resolved with meta heuristic algorithm termed Accelerated Cuckoo Search Based Job Sharing Algorithm, and it is attained to be extremely precise and forceful as compared to prevailing algorithms found, as the algorithm considerably minimizes total executing time along with computational time, and percentage of rate at which deadline is fulfilled is also compared to existing algorithms. The results of the anticipated method outperform the prevailing methods in terms of effectual job sharing; transmission speed, Bandwidth used, execution time of a job, transmission value, through put value, buffering overhead and reduced waiting time. As future work extension, diverse options for obtaining initial pool of tasks can be considered as it provides significant impact over performance of heuristic algorithm. The work is also likely to be experimented with diverse optimization strategies like genetic algorithms and compare their recital with Cuckoo Search Algorithm. Furthermore, this approach is developed to work in multi-mode cloudlet, and it can be generated to work amongst huge data centres and accounting network bandwidth. At last, it is targeted to implement the process in workflow engine, hence that it can be cast off for deploying applications in real time environments.

REFERENCES

[1]. H. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” in Wireless Commun. and Mobile Comput., 2011.

[2]. A. Khan and K. Ahirwar, “Mobile cloud computing as a future of mobile multimedia database,” in International Journal of Computer Science and Communication, Vol. 2, No. 1, January-June 2011, PP. 219-221.

[3]. D.Huang, P.Wang and D.Niyato, “A dynamic offloading algorithm for mobile computing”, Published in: IEEE Transactions

on Wireless Communications Volume: 11 , Issue: 6 , June 2012 11(6), 1991–1995 (2012)

[4]. K.Liu, J.Peng, X.Zhang and Z.Huang, "A combinatorial optimization for energy-efficient mobile cloud offloading over cellular networks", IEEE Global Communications Conference (2016), DOI: 10.1109/GLOCOM.2016.7841488

[5]. J.Cheng, Y.Shi, B.Bai and W .Chen, ,"Computation offloading in cloud-RAN based mobile cloud computing system", IEEE International Conference on Communications (2016), DOI: 10.1109/ICC.2016.7511367

[6]. S.Zhang, L.Pan , L.Wu , S.Liu and X.Meng, "Dynamic-Priority based Profit-Driven Scheduling in Mobile Cloud Computing", Proceedings of the IEEE 21st International Conference on Computer Supported Cooperative Work in Design, 2017 DOI: 10.1109/CSCWD.2017.8066710.

[7]. X.Lin, , Y. Wang , Q.Xie and M. Pedram," Energy and Performance-Aware TaskScheduling in a Mobile Cloud Computing Environment", IEEE International Conference on Cloud Computing, 2014,DOI: 10.1109/CLOUD.2014.35

[8]. Y.Geng, , Y.Yang and G.Cao "Energy-Efficient Computation Offloading for Multicore-Based Mobile Devices", IEEE INFOCOM 2018 - IEEE Conference on Computer Communications DOI: 10.1109/INFOCOM.2018.8485875

[9]. C. Tseng, F.Tseng,Y.Yang, C. Liu, and L.Chou "Task Scheduling for Edge Computing with Agile VNFs On-Demand Service Model toward 5G and Beyond", Hindawi Wireless Communications and Mobile Computing Volume 2018

[10]. L.Liu, D.Qi, N.Zhou and Y.Wu, "A Task Scheduling Algorithm Based on Classification Mining in Fog Computing Environment", Hindawi Wireless Communications and Mobile Computing Volume 2018.

[11]. M. Joshi, A.Tornekar and R. Sawkar " An Improved Scheduling Algorithm For Task Offloading On Cloud", International Journal of Technical Research and Applications, Volume 4, Issue 3 , May-June, 2016), PP. 272-275

[12]. L.Tianze, W.Muqing , Z.Min and L.Wenxing , "An Overhead-Optimizing Task Scheduling Strategy for Ad-hoc Based Mobile Edge Computing", IEEE 2017.DOI: 10.1109/ACCESS.2017.2678102

[13]. O.Chabbouh, S.Rejeb,Z.Choukair andN.Agoulmine "A strategy for joint service offloading and scheduling in heterogeneous cloud radio access networks" EURASIP Journal on Wireless Communications and Networking (2017) 2017:196 DOI 10.1186/s13638-017-0978-0

[14]. H.Qian and D.Andresen, "An Energy-saving Task Scheduler for Mobile Devices", IEEE/ACIS 14th International Conference on Computer and Information Science, Electronic DOI: 10.1109/ICIS.2015.7166631

[15]. S.Banerjee, S.Mukherjee andU. Biswas "An Approach towards Development of a New Cloudlet Allocation Policy with Dynamic Time Quantum", Automatic Control and Computer Sciences May 2018, Volume 52, Issue 3, pp 208–219

[16]. Y. C. Lee and A. Y. Zomaya, "Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions," IEEE Transactions on Paralleland Distributed Systems, vol. 22, no. 8, pp. 1374–1381, 2011.

[17]. Y. Zhu and V. Reddi, "High-Performance and Energy-Efficient Mobile Web Browsing on Big/Little systems," IEEE 19th International Symposium on High Performance Computer Architecture 2013. DOI: 10.1109/HPCA.2013.6522303

[18]. W. Hu and G. Cao, "Energy Optimization Through Traffic Aggregation in Wireless Networks," IEEE Conference on Computer Communications 2014. DOI: 10.1109/INFOCOM.2014.6848020

[19]. L. Tong and W. Gao, "Application-Aware Traffic Scheduling for Workload Offloading in Mobile Clouds," IEEE International Conference on Computer Communications ,2016. DOI: 10.1109/INFOCOM.2016.7524520

[20]. W. Zhang, Y. Wen, and D. O. Wu, "Energy-Efficient Scheduling Policy for Collaborative Execution in Mobile Cloud Computing," Proceedings IEEE INFOCO 2013. DOI: 10.1109/INFCOM.2013.6566761

[21]. B.G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution between Mobile Device and Cloud,"

in EuroSys '11 Proceedings of the sixth conference on Computer systems Pages 301-314

[22]. Z.Wang, Q.Zhao, F.Xu, H.Dai, and Y.Zhang, “Detection Performance of Packet Arrival under Downclocking for Mobile Edge Computing,” Wireless Communications and Mobile Computing, vol. 2018,

[23]. N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile Edge Computing: A Survey,” Published in: IEEE Internet of Things Journal (Volume: 5 , Issue: 1 , Feb. 2018) Page(s): 450 - 465DOI: 10.1109/JIOT.2017.2750180

[24]. L. Zhao, W. Sun, Y. Shi, and J. Liu, “Optimal Placement of Cloudlets for Access Delay Minimization in SDN-Based Internet of Things Networks,” IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1334–1344, 2018.