Resource Allocation for Heterogeneous Cloud Computing using Weighted Fair-Share Queues

K. Naveen Kumar
Computer Science and Engineering
IIIT Vadodara, Gujarat
India
201451074@iiitvadodara.ac.in

Reshmi Mitra
Mathematics & Computer Science
Webster University, St. Louis
USA
reshmimitra25@webster.edu

***Abstract*— On-demand resource provisioning is based on automated approaches for resource pooling and elasticity on the cloud service provider (CSP) side. The infrastructure services must be adapted dynamically to accommodate customer demands and yet, operate within offerings of the CSP. Although multiple approaches for homogeneous clouds are available, more realistic platforms based on heterogeneous resources and virtual machines (VMs) present unique challenges. Our resource management algorithm allocates memory, network and computational resources to heterogeneous VMs, in order to provide customized fine-grained control for the scalable capacity planning of datacenters. Weighted fair-share (WFS) queues: high, medium and low are used to classify the incoming jobs in buckets of appropriate length based on priority. The highest priority jobs with aggressive deadlines are allowed to progress at a similar pace using round-robin scheduling, while lowest priority jobs are allocated on Low Queue with First-Come-First-Serve (FCFS) scheduling. The proposed algorithm performs better on throughput related metrics: number of instructions executed (30% more), turn-around and waiting times (on an average of about 10% less) w.r.t. standard policies such as shortest job first (SJF) and FCFS.**

Keywords: dynamic control, resource allocation, provisioning, heterogeneous workload, throughput, scalability.

I. INTRODUCTION

Modeling heterogeneous workloads running on heterogeneous platforms is a challenging task [1], because it cannot be addressed with the existing resource allocation policies. Ideally, throughput should increase with the rise in the hardware capacity. However, with the law of diminishing returns, it may saturate at some point. Moreover, there is a substantial uncertainty over workload behavior on the different nodes in the datacenter, because the cloud engineer may not have a full understanding of their performance [2]. To accommodate contradictory application requirements (such as deadline, priority, among others), it now becomes a multi-parameter optimization problem. The cloud engineer needs to develop novel heuristics and solutions, which is a digression from the previous "one-solution-fits-all" strategy [3].

Substantial work has been done on resource management policies for the homogeneous clouds, however heterogeneous resources is a more realistic setting. After the hardware breaks down in homogeneous clouds with the progression of time, it becomes difficult to replace the nodes with the exact same hardware through the life-cycle of the entire cloud. For simplifying the management mechanism, the cloud engineer may end up under-utilizing its full capacity. Our work addresses this issue of achieving throughput scalability by identifying customized resource allocation solutions for pre-emptive workloads with priority and deadline. The primary objective is to develop and evaluate resource management algorithm for heterogeneous cloud computing systems with multi-criteria requirements. This work proposes a hierarchical round-robin based algorithm to allocate memory, network and computational resources for virtual machines (VMs) running on computing systems with diverse capacities.

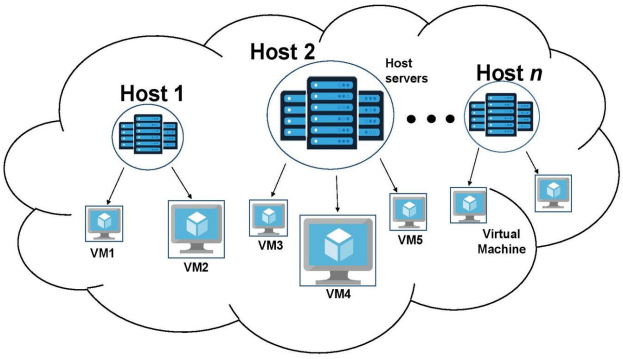

Fig. 1: Abstraction of heterogeneous resources and host machines

Our Weighted Fair-Share (WFS) algorithm uses fast, customized and fine-grained control for scalable capacity planning of datacenters by scheduling jobs among three-leveled queues: high, medium and low abbreviated as HQ, MQ, and LQ. To cater to a dynamic workload, the size of each queue can be varied using regulating parameters which are based on job properties - priority and deadline. Within each queue, the scheduling policy for HQ and MQ is round-robin and the LQ is on the First-Come-First-Serve (FCFS) basis. In comparison to the high priority and aggressive deadlines jobs scheduled on HQ and MQ, jobs with low priority with lenient deadlines are demoted to LQ. Hence, the former can complete starvation-free execution with low turn-around time and low wait times. The three main parts of the proposed algorithm are: (a) division of tasks into HQ, MQ and LQ

(b) creation of VMs and (c) allocating tasks to VMs. This algorithm is for the data-center consolidation without VM migration. This is a NP-hard problem involving optimization in VM creation and efficiently allocating resources to tasks.

For the performance evaluation, the workload characteristics includes number of instructions, arrival time, burst time, deadline, priority among others. The proposed algorithm demonstrates an overall performance gain on six diverse workloads - overloaded (with competing deadlines), underloaded and various combination of light- or heavy-weight tasks. It performs better on all throughput related metrics - the number of instructions executed, turn-around and waiting times w.r.t. standard algorithms such as shortest job first (SJF) and FCFS. The main contributions of this research are as follows:

- Identify an approach for throughput scalability in data-center with heterogeneous host machines.
- Design and validation of resource allocation algorithm for the heterogeneous cloud computing system with six different types of workload.
- Performance bottleneck deep-dives on workload cases where existing algorithms are under-performing.

There are four main sections in this paper. The related work about resource allocation problem is presented in Section 2. The entire design of the proposed model framework is explained in Section 3. The model results and discussions are presented in Section 4. The concluding remarks are presented in the last section with the summary of the main contributions, and the future work.

II. RELATED WORK

The related work focuses on variable resource management, utility optimization, and minimalistic VM design. The dynamic resource allocation problem for system capacity and application-level performance has received academic and commercial interest for last ten years [4], [5], [6]. Characterization of *heterogeneous* web traffic in cloud computing received some interest [7], however, there has been limited study on how to solve the dynamic capacity provisioning problem in heterogeneous datacenters [3].

Seminal work on cloud computing research challenges [2] highlights the importance of automatic resource provisioning and development of effective monitoring for allocation decisions in order to achieve usage optimization. The objective of the dynamic resource allocation for VMs [8] is to minimize the skewness metric to maintain the resource utilization in servers. They are using a exponentially weighted moving average value to gauge the real time workload variation. More recently the survey article [9] raised some key issues such as achieving predictable performance on a local as well as global scale, and most importantly measurement studies for scalable resource management. The recent trend has also moved to energy-aware allocation [10]. However, our understanding is that heterogeneous workloads on heterogeneous resources is still an under-explored area and green cloud computing is beyond the scope of current work. Hence, we have intentionally not included that body of work in our review.

Early works such as [1] have provided insights about heterogeneous nature of Google cluster traces and resources types (core, memory). They have postulated that predictable job patterns can help resource scheduling efforts, which has better performance outcomes as compared to the traditional slot- or core-based scheduling. Overall, our results corroborate with their findings that multi-layered job schedulers will support such realistic workloads and can guide future scheduler design.

The cost function based approach [3] presented early solutions for this *heterogeneous (both workload and resource)* problem with the goal to minimize the total cost associated with resources, reconfiguration, performance, and capacity penalty. Their technique involves containers as logical task allocation to help the controller in making resource allotment decisions. They have solved the dynamic service placement problem using K-means algorithm to divide tasks into classes. Our hierarchical queue-based approach provides fine-grained control for fast switching between the high priority tasks with aggressive deadlines. In addition, we have considered six different types of workloads in comparison to their single MapReduce. Prior work such as SLA-based resource scheduling [6] have defined the parameters in the user requirements.

Another cost function based approach [11] proposes heuristics to find a near-optimal solution for optimization problem of multiple VMs for processor allocation without considering other resource requirements such as memory or workload requirements such as deadlines, arrival time or priority. Their algorithm is essentially searching for an optimal solution based on the utility value to reach to maximum optimum level for the resource allocation without proper accommodations for newer VM configurations.

Earlier resource allocation work [4] has characterized the overheads associated with the computational and transactional workloads for virtual containers. The cost-function approach [5] uses a utility function for priority-based resource scheduling for CPU and memory in VMs with a small-scale test-bed that fails to capture the complete scale of heterogeneity. The uneven utilization of a server by mixed workloads is represented by resource "skewness" [12], and their goal is to avoid overloading the VMs. In case, the metric reaches a particular threshold value, it forces the VMs to migrate to ease the load. Dynamic and Integrated Resource Scheduling (DAIRS) [13] considers CPU, memory and network bandwidth to integrate physical and VMs to measuring total (load) imbalance level in cloud datacenter including average imbalance level of each server.

Cloud Resource management [5] uses utility function and proposes a set of rules for the framework. However, the abstractions for the workload requirements are theoretical in nature and does not justify the practical task constraints such as deadlines and priorities. Priority-based VM infrastructure allocation with load balancing enables efficient resource utilization [14]. Other load balancing works for VM allocation

on cloud [15], [16] used genetic algorithm based approaches. However, the focus of our work is not on load-balancing but on designing effective algorithms for long-running heterogeneous jobs, and hence is beyond the scope of our work. In the dynamic resource allocation algorithms [17] in heterogeneous multicloud system uses using min-min and list scheduling algorithm for preemptable tasks. However, there is contigency about reliable feedback information to the scheduler.

Earlier a study of two open sources is given by [18], where virtual infrastructure managers called OpenNebula and Haize explains the dissimilarities between existing features which rely on a false assumption of immediate resource provisioning with practically infinite resources and the importance of prioritizing, queued, pre-reserved, requests for resources to be deployed on external clouds. Our proposed algorithm is based on the concept of WFS used in computer networks to manage the link capacity for the packet flow. We have extended the algorithm proposed in [19], where they have used three-layered scheduling policies for the tasks based on the deadline and cost. Their time quantum is static, but ensures that frequent task switching happens in the high queue. Their performance metric is the turnaround time and cost of available resources without any concrete performance evaluation.

We have included priority, deadlines, variable time quantum, creation of heterogeneous VMs using multiple strategies and allocation of tasks to VMs using the best-fit approach. Our performance metric is turnaround time, waiting time, the percentage of task execution w.r.t. assigned. We considered six distinct test cases such as heavy-weight, light-weight, and combinations of intermediary tasks to demonstrate performance improvement in comparison to standard scheduling schemes (FCFS and SJF). We have considered 6 possible workloads with performance metrics including Turnaround time, Waiting time, percentage of instructions executed with the three different strategies for VM creation. Our results show that WFs achieved better overall results as compared to SJF and FCFS by putting the best-fit policy to its maximum use.

III. MULTI-LEVEL QUEUE FOR HETEROGENEOUS DATA-CENTER

A. Motivation for Proposed Approach

Although throughput increases with the increase in the hardware resources in ideal condition, performance saturation sets in due to the law of diminishing returns. Another important point is that one-solution-fits-all is an unsuitable strategy, especially for large workloads. Hence, to identify a better strategy, it is important to understand throughput with respect to application and hardware scalability. Fig. 2 shows this relationship by breaking into four different categories. The *x*-axis represents the task weightage and the *y*-axis shows the hardware capacity (small or large clouds). The most interesting part is the 1^{st} Quadrant, which needs some customized planning as heavy-weight tasks are running on large clouds. The cloud in the 2^{nd} Quadrant will be under-utilized since it is catering to low load. Obviously, light-weight tasks on small clouds are not of much interest and hence, has been shaded out (3^{rd} Quadrant). In the 4^{th} Quadrant, the large number of tasks cannot be sustained on the small clouds and will cause overload situation which is undesirable. In summary, careful capacity planning is required for heavy-weight tasks (1^{st} and 4^{th} Quadrants) as shown in Fig. 2, although the approaches may vary.

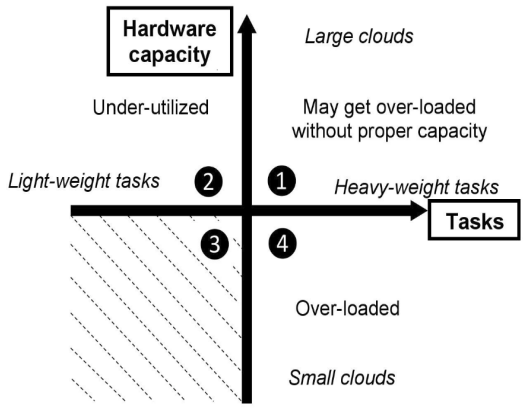

Fig. 2: Abstraction of task requirement matching with the cloud capacity and the motivation for the design of algorithm to handle datacenter consolidation.

B. Weighted Fair-Share Queues - Proposed Approach

The proposed approach is based on the divide-and-conquer strategy. The tasks are divided into queues undergoes sorting before allocating resources through the various scheduling policies in a best-fit manner.

Overall Inputs: Task properties and their respective resource requirements are as follows:

- IC: Instruction count
- *RAM*: RAM requirement in GB
- *StrSpace*: Storage space requirement in GB
- *AT*: Arrival time in seconds
- *BT*: Burst time in seconds
- *Pr*: Priority
- *Dl*: Deadline in seconds

Host resource specification is listed below:

- *MIPS*: Million Instructions Per Second
- *RAM*: RAM availability in GB
- *StrSpace*: Storage space availability in GB

Overall Outputs :

- *ATT*: Average Turnaround time
- *AWT*: Average Waiting time
- Number of instructions executed per testcase

The algorithm is divided into three stages:

1) Stage 1 - Division into task queues: We are generating the three queues as output in the first stage with length calculations according to equation (1), (2) and (3). It takes the entire task array information including priority, arrival time and deadline as the input. Tasks are divided into high-queue (HQ), mid-queue (MQ) and low-queue (LQ) based

on the priority of the tasks assigned as shown in Fig. 3 to represent the resource constraint situation. The tasks in HQ and MQ are executed in round-robin fashion with a time quantum of TQ_H and TQ_M respectively. The calculation of the time quantum is shown later in this section. The tasks with the lowest priority are executed in LQ based on FCFS fashion. The number of tasks is assumed to be larger than the number of hosts (1^{st} and 4^{th} quadrants in Fig. 2). The details of this stage are shown in algorithm 1.

Inputs are as follows:

Pr[ ]: Priority, **AT[ ]**: Arrival time, **Dl[ ]**: Deadline array

Outputs are as follows:

HQ, MQ, LQ, TQ_H, TQ_M

- N_{HQ}, N_{MQ}, N_{LQ} is the number of tasks in HQ, MQ, LQ respectively. The first N_{HQ} number of tasks in *Sort_Tasks* array are placed in HQ, next N_{MQ} are placed in MQ and the remaining in LQ.

$$N_{HQ} = \lfloor \frac{n}{3} \rfloor + \lambda \quad (1)$$

$$N_{MQ} = \lfloor \frac{(n - N_{HQ})}{2} \rfloor \quad (2)$$

$$N_{LQ} = \lfloor n - (N_{HQ} + N_{MQ}) \rfloor \quad (3)$$

- λ is a positive integer to regulate the number of tasks in HQ, MQ, LQ and handles load balancing.
- *n* is the total number of tasks.
- The time quantum for HQ and MQ round-robin are calculated with the following equations:

$$TQ_H = \lfloor \frac{max(DL(T''_{i,HQ}))}{2^{\alpha_1}} \rfloor \quad (4)$$

$$TQ_M = \lfloor \frac{max(DL(T''_{i,MQ}))}{2^{\alpha_2}} \rfloor \quad (5)$$

where,

- $max(DL(T''_{i,HQ}))$ and $max(DL(T''_{i,MQ}))$ are the deadlines in HQ and MQ respectively.
- α_1 and α_2 are the positive integers to regulate time quantum and $\alpha_1 > \alpha_2$ to ensure $TQ_H < TQ_M$.
- AN, AR and AS are the average of instruction count, RAM and Storage Space of HQ tasks, respectively and used in average strategy (Stage 2).

2) Stage 2 - Creation and Allocation of VMs: VMs are created from the given host resources in this stage. They are based on three different strategies namely average, minimum and default.

Inputs are as follows:

HDict: Dictionary for host information with host-id as keys and specifications such as RAM, storage space and MIPS information as values

RAM: RAM in GB

StrSpace: Storage Space in GB

InstrCount: Number of instructions for the task

Outputs are as follows:

VMDict: Dictionary for VM information with VM id as keys and specifications such as RAM, storage space and MIPS information as values

Average Strategy: This is the best case scenario, wherein averaging the specifications of tasks and creating VMs will enable maximum use of resources with less wastage. In case it fails, the other two strategies can be considered. To first execute HQ, we average the specifications of the tasks in HQ and create VM_1 to build the other VMs based on the VM_1 as shown in Table I where *iRAM*, *iStr*, *iInst* define the average of RAM, instruction count and storage space of HQ tasks respectively. HRAM_i, HMIPS_i, HStrspace_i, represent the RAM, MIPS and storage space specification for the i^{th} machine.

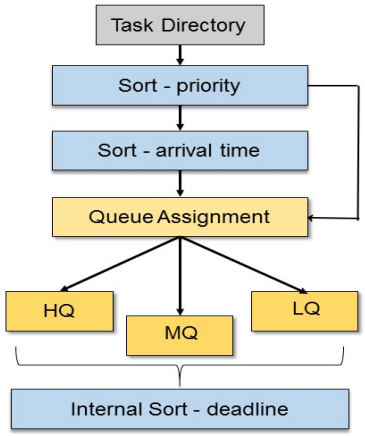

Fig. 3: Task division into multi-level queues based on priority, arrival time and deadline

Stage 1: Division of Tasks into High Queue (***HQ***), Mid Queue (***MQ***), Low Queue (***LQ***)

Input: ***Pr***[ ], ***AT***[ ], ***Dl***[ ]
Output: ***HQ***[ ], ***MQ***[ ], ***LQ***[ ], $\boldsymbol{TQ}_H$, $\boldsymbol{TQ}_M$

$Sort\_Task[] \leftarrow [T'_1, T'_2, ..., T'_n]$
// Insertion sort with decreasing order of priority.
$\boldsymbol{Pr}[T'_1] \leftarrow$ highest priority task ; i $\leftarrow 1$, $j \leftarrow 2$
while $i < n$ *and* $j < n$ **do**
  if $i \neq j$ **then**
    if $\boldsymbol{Pr}[T'_i] == \boldsymbol{Pr}[T'_j]$ **then**
      // Insertion sort with increasing order of arrival time.
      $\boldsymbol{AT}[T'_1] \leftarrow$ lowest arrival time task
  i++, j++
+6Calculate N_{HQ}, N_{MQ} & N_{LQ} using equations (1), (2) and (3)
// Insertion sort (internally) with increasing order of deadlines
$\boldsymbol{HQ}[\ ] \leftarrow [T''_1, T''_2, ..., T''_{N_{HQ}}]$
if $\boldsymbol{Dl}[T''_i] == \boldsymbol{Dl}[T''_j]$ **then**
  if $\boldsymbol{Pr}[T''_i] == \boldsymbol{Pr}[T''_j]$ **then**
    $\boldsymbol{HQ}$[i] $\leftarrow$ lowest arrival time task
  else
    $\boldsymbol{HQ}$[i] $\leftarrow$ highest priority task
else
  $\boldsymbol{Dl}[T''_1] \leftarrow$ earliest deadline task
Similarly $\boldsymbol{MQ} \leftarrow [T''_1, .., T''_{N_{MQ}}]$ & $\boldsymbol{LQ} \leftarrow [T''_1, .., T''_{N_{LQ}}]$
Calculate $\boldsymbol{TQ}_H$ & $\boldsymbol{TQ}_M$ using equations (4) and (5)

TABLE I: VM Creation using different strategies

	RAM	**StrSpace**	**MIPS**
VM_1	iRAM	iStr	iInst
VM_2	iRAM+HRAM$_2$/2	iStr+HStrSpace$_2$/2	iInst+HMIPS$_2$/2
VM_3	iRAM+HRAM$_3$/2	iStr+HStrSpace$_3$/2	iInst+HMIPS$_3$/2
.	.	.	.
VM_n	iRAM+HRAM$_n$/2	iStr+HStrSpace$_n$/2	iInst+HMIPS$_n$/2

Minimum Strategy: In case the average strategy fails in VM creation, we will consider the minimum of the task specifications of HQ tasks and create VM say VM_1.

Stage 2: Creation and allocation of VMs based on host machine capacities

Input: ***HDict***{ }, ***HQ***[ ], ***MQ***[ ], ***LQ***[ ]
Output: ***VMDict***

for $i \leftarrow 1$ **to** $HLength$ **do**
  // Average Strategy
  if *i == 1* **then**
    [AN, AR, AS] ← Avg(HQ [InstrCount, RAM, StrSpace])
  else
    // other VMs ← VM_1 values plus half the current host values as shown in **Table I**
  if *[AN, AR, AS] <= host[i]* **then**
    VMDict[i] ← [AN,AR,AS]
    //VMId allocated to respective hosts
    update ***HDict*** with new entry
  else
    //Average strategy failed, use Minimum strategy
    if *i == 1* **then**
      [MN, MR, MS] ← Min (***HQ*** [InstrCount, RAM, StrSpace])
      if *[MN, MR, MS] <= host[i]* **then**
        VMDict[i] ← [MN, MR, MS]
        update ***HDict*** with new entry
      else
        [DN, DR, DS] ← (host[i])/2
        VMDict[i] ← [DN, DR, DS]
        update ***HDict*** with new entry
    else
      //Average strategy failed for other VMs, using minimum strategy in **Table I**
      if $[MN', MR', MS'] <=$ *Host[i]* **then**
        VMDict[i] ← $[MN', MR', MS']$
        update ***HDict*** with new entry
      else
        //as shown in **Table I**
        VMDict[i] ← $[DN', DR', DS']$
        update ***HDict*** with new entry

We will allocate the other VMs based on initial VM_1 allowing least valued tasks to be executed in the end. The strategy is shown in Table I where *iRAM*, *iStr*, *iInst* define the minimum of RAM, instruction count and storage space of HQ tasks respectively.

Default Strategy: In case both the above strategies fail, consider half of host specifications to create VM_1 as default setting. Allocating the other VMs based on VM_1 will ensure that this strategy never fails and allows resource usage in the worst case scenario. The strategy is shown in Table I where *iRAM*, *iStr*, *iInst* define the half of RAM, MIPS and storage

Stage 3: Task Allocation to created VMs

Input: ***VMDict***{ }, ***HDict***{ }, ***TQ***$_H$, ***TQ***$_M$, ***HQ***[ ], ***MQ***[ ], ***LQ***[ ]
Output: ***TTAT***[ ], ***TWT***[ ], ***ExecPercent***[ ]

TDict(Key ← VMId; values ← AllocTask[ ])
function Task_to_VM (Qname[ ], TQ):
i ← 1
out_dict : {TTAT[ ], TWT[ ], ExecPercent[ ]}
while i ≤ *TLength* **do**
for $j \leftarrow 1$ **to** *VMLength* **do**
  if *[TRAM[i], TStrSpace[i], TInstrCount[i]] ≤ [V_RAM[j], V_StrSpace[j], V_MIPS[j]* **then**
    TDict[VM_Id[j]] ← HQ[i]
    Roundrobin(Qname[ ], TQ_H, out_dict)
    //In case of LQ execution
    FCFS(LQ[ ], out_dict)
    break
  //Creation of new VMs
  else
    ExtList[ ] ← [TInstrCount[i], TRAM[i], TStrSpace[i]] //ExtList contains specifications of Extra VMs to be created
    for $k \leftarrow 1$ **to** $Hlength$ **do**
    if *ExtList[ ] ≤ [HMIPS[k], HRAM[i], HStrSpace[i]]* **then**
      next ← VMLength + 1
      VmDict[next] ← ExtList[ ]
      HRAM[k] ← HRAM[k] - VRAM[next]
      HStrSpace[k] ← HSize[k] - VMStrSpace[next]
      TDict[Vms[next]] ← HQ[i]
      Roundrobin(Qname[ ], out_dict)
      //In case of LQ execution
      FCFS(LQ[ ], out_dict)
    else
      Throw **exception**
//Extra VMs created in this stage are destroyed and values are added back to their respective Hosts
for $i \leftarrow 1$ **to** $ExtraVmLength$ **do**
  HRAM[i] ← HRAM[i] + ExtraVRAM[i]
  HSize[i] ← HSize[i] + ExtraVStrSpace[i]
Task_to_VM(**HQ**[ ], ***TQ***$_H$) //Function call
Task_to_VM(***MQ***[ ], ***TQ***$_M$)
//Similarly LQ Tasks executed in FCFS
Task_to_VM(LQ[ ])

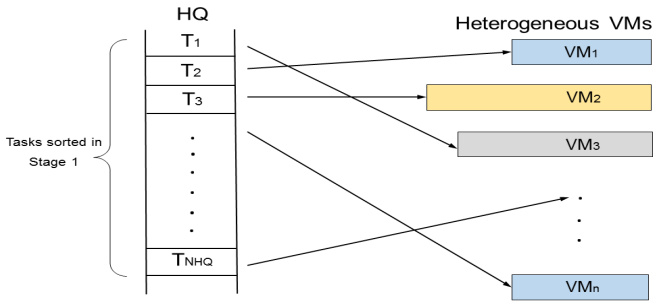

Fig. 4: Task Allocation to created VMs

space of $host_1$ respectively. The output from the previous stage (task arrays) is the input to this stage. The details are shown in Algorithm 2.

3) Stage 3 - Task Allocation to created VMs: This is the final stage where tasks are allocated to VMs in a best-fit policy and executed in round-robin and FCFS fashion as shown in Fig. 4.

Inputs: HQ, MQ, LQ with tasks with time quanta calculated as per equations (4) and (5).

Outputs: TTAT, TWT and number of instructions executed per task. These metrics are calculated as per equations (6), (7), (13) respectively. The details are shown in Algorithm 3.

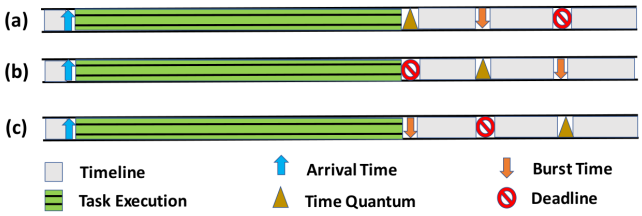

Fig. 5: Various task timeline possibilities: (a) time quantum arrives before burst time and deadline of tasks, (b) deadline arrives before burst time and task time quantum; and (c) burst time arrives before deadline and task time quantum

The performance metrics under study are as follows:

- TCT: Task completion time
- TTAT: Task Turnaround time and is calculated using the following equation:

$$TTAT = TCT - TAT \tag{6}$$

- TWT: Task Waiting time and is calculated using the following equation:

$$TWT = TTAT - TET \tag{7}$$

- TET: Task Execution time calculation for the different HQ and MQ possibilities in Fig. 5 (a), (b), (c). The equations are shown below:

$$TET = TQ - TAT \tag{8}$$

$$TET = TDL - TAT \tag{9}$$

$$TET = TBT - TAT \tag{10}$$

TET is calculated in LQ using:

$$TET = TBT - TAT \tag{11}$$

$$TET = TDL - TAT \tag{12}$$

- IE: Number of instructions executed and is calculated using the following equation:

$$IE = \frac{TET}{TBT} * IC \tag{13}$$

- PIE: Percentage of instructions executed and is calculated using the following equation:

$$PICE = \frac{No.of ICexecuted}{IC} * 100 \tag{14}$$

IV. Performance Evaluation

A. Experimental Set-up

The experimental setup evaluates simulation of practical task scenarios outlined in Fig. 2, with six diverse workloads viz., overloaded (with competing deadlines), underloaded and various combination of number of light- or heavy-weight tasks as listed below:

(a) *Testcase 1*: Heavy-weight tasks with competing deadlines
(b) *Testcase 2A*: Small number of tasks with large resource requirements
(c) *Testcase 2B*: Large number of tasks with small resource requirements
Note: The overall instructions for the above two test cases are similar, but they cover two different types of workload
(d) *Testcase 3A*: Small number of tasks with small resource requirements
(e) *Testcase 3B*: a Large number of tasks with large resource requirements
(f) *Testcase 4*: Light-weight tasks

The above tests as summarized in Table II specifies the number of instructions, RAM requirement, arrival time, burst time, deadline and priority for the particular testcase. For example, the heavy-weight testcase 1 has high resource and task requirements - instruction count (800 to 1000), RAM (800 to 1000) and burst time (1 to 100), respectively. In comparison light-weight case 4 has much smaller instruction count (100 to 300), RAM (200 to 400) and burst time (1 to 3), respectively. The hosts are determined by their MIPS, RAM, and storage space. The performance metrics are calculated using WFS simulator written in Python 2.7 for ATT (average turn-around time), AWT (average waiting time) and the number of instructions executed per task as outlined in equations (6) to (14).

B. Result Discussion

The highlights of the results are presented below :

- As shown in Fig. 6 and Fig. 7, highest performance gain in terms of TTAT and TWT are with the test case 4 and 2A with light-weight tasks and small number of tasks with large resource requirements, respectively. Allocating light-weight tasks (testcase 4) is relatively easier and faster, because of the higher

TABLE II: Testcase Parameters: Details about value range for the different testcases used in performance evaluation

	Instruction Count Range	RAM Range (GB)	Storage Space Range (GB)	ArrivalTime Range (sec)	BurstTime Range(Sec)	Deadline Range(sec)	Priority Range
Testcase1	[800, 1000]	[6, 10]	[800, 1000]	[0, 3]	[1, 10]	[5, 8]	[1, 4]
Testcase2A	[1000, 1500]	[10, 15]	[900, 1000]	[0, 3]	[8, 15]	[9, 13]	[1, 4]
Testcase2B	[100, 300]	[1, 4]	[100, 400]	[0, 3]	[1, 4]	[1, 5]	[1, 4]
Testcase3A	[100, 300]	[1, 4]	[100, 400]	[0, 3]	[1, 4]	[1, 5]	[1, 4]
Testcase3B	[1000, 1500]	[10, 15]	[900, 1000]	[0, 3]	[8, 15]	[9, 13]	[1, 4]
Testcase4	[100, 300]	[1, 3]	[200, 400]	[0, 3]	[1, 3]	[3, 4]	[1, 4]

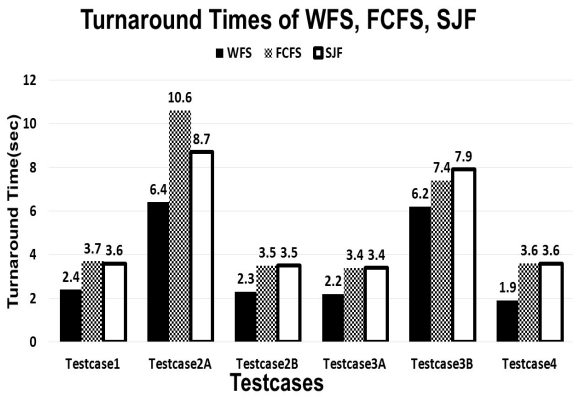

Fig. 6: Comparison plots for turn-around time for WFS, FCFS, SJF with the six testcases

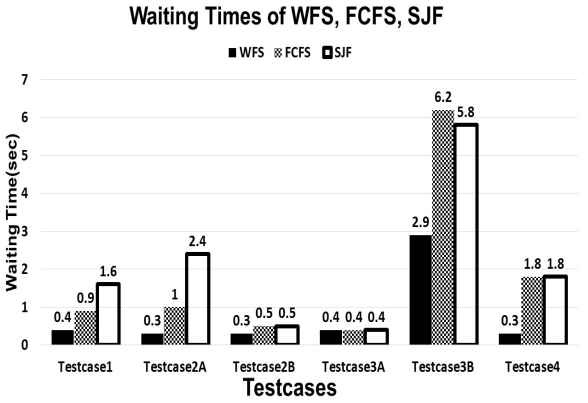

Fig. 7: Comparison plots for waiting times in WFS, FCFS, SJF

availability of resource. Similarly, in low demand testcase 2A, the resources are sufficient enough to fulfill requirements for the small number of tasks. The performance gain in both the cases can be given by

$$Performance\,gain = \frac{current - previous}{current} * 100 \quad (15)$$

To show sample calculation for testcase 4 using equation (15), the performance gain is given by ((3.6-1.9)/3.6)*100 = 50%, which is halfway reduction in terms of TTAT.

- In comparison to FCFS and SJF, WFS shows a minimum gain for TTAT and TWT in testcase 2B and 3A, as they both comprise of small resource requirements. In other words, customized allocation WFS does not translate into effective gains for low demands.
- Most importantly, maximum execution for testcase 1 shows best results as compared to other test cases as

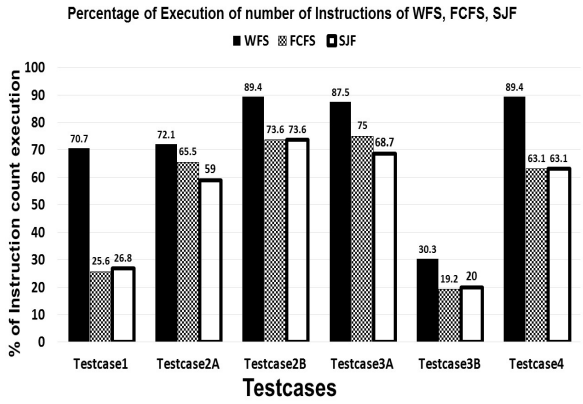

Fig. 8: Comparison plots for execution percentage in WFS, FCFS, SJF

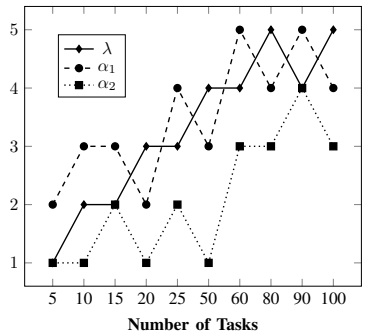

Fig. 9: Characteristics of regulating parameters such as $\boldsymbol{\lambda}$, $\boldsymbol{\alpha_1}$, $\boldsymbol{\alpha_2}$ values for scaled tasks in respective testcases resulting in best performance

shown in Fig. 8. This gain is due to WFS allowing partial execution of tasks in contrast to SJF and FCFS.

V. CONCLUSION

Instantiating VMs and job assignment on heterogeneous cloud resources is a multi-parameter optimization problem based on job priority, deadline and arrival times. Instead of the prior "one-solution-fits-all" approach for homogeneous resources, we have shown that there is a strong need for a novel customized scheduling policy based on the task characteristics and capacity of the cloud resources. In our work, we have developed a resource allocation algorithm based on *weighted fair-share queues* for the on-demand resource provisioning problem for the heterogeneous workload in the cloud. By using fair-share queues our algorithm allows (multiple) high priority jobs with aggressive deadlines to advance in a time-sharing manner, while demoting the lower priority jobs. Our algorithm outperforms the standard scheduling policies on six different workload scenarios ranging from highly overloaded to under-loaded jobs.

Our solution performs better in comparison to the other standard policies because of multiple reasons. First, we use a three-leveled queue with a tuning parameter for fine-grained control of the high priority jobs. There are additional 'adjuster knobs' (λ along with α_1 and α_2) to customize the queue lengths and time sharing of resources based on the job pattern (priority and deadline). Secondly, a combination of strategies such as average, minimum, default in WFS Stage 2, rather than a static rule ensures better resource usage. It is hard to design a theoretical model for such a multi-level customized scheduling policy. However, our result for the six practical workload type demonstrate the success of our approach in comparison to the standard scheduling policies such as FCFS and SJF.

We plan to extend this work by adding a policy library that can save past assignments (task type, VM resources) to reuse them based on incoming job pattern. This will save time from calculating the VM configurations on real-time and thus, the system can achieve better (task) throughput.We also want to extend our work to fog computing problems with IoT-based devices which are ideal test situations for the heterogeneous cloud problems.

ACKNOWLEDGMENT

This work was done when K. Naveen Kumar and Reshmi Mitra were in Indian Institute of Information Technology Vadodara, Gujarat, India.

REFERENCES

[1] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis,"*Proc. ACM Symp. Cloud Computing, 2012*, October 14-17, 2012.

[2] Q. Zhang, L. Cheng, R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Springer Journal of Internet Services and Applications*, 2010.

[3] Q. Zhang, M.F. Zhani, R. Boutaba, J.L. Hellerstein, "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Transactions on Cloud Computing*, Vol. 2, No. 1, January-March 2014.

[4] Z. Wang, X. Zhu, P. Padala, S. Singhal, "Capacity and Performance Overhead in Dynamic Resource Allocation to Virtual Containers," *IEEE, 10th IFIP/IEEE International Symposium on Integrated Network Management*,DOI: 10.1109/INM.2007.374779, 2007.

[5] Y. Song, Y. Li, H. Wang, Y. Zhang, B. Feng, H. Zang, Y. Sun, "A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing," *International Conference on High-Performance Computing*, vol. 5374, 2008.

[6] L. Wu, S.K. Garg and R. Buyya, "SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011

[7] A.K. Mishra, J.L. Hellerstein, W. Cirne, and C.R. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 37, pp. 34-41, Mar. 2010.

[8] Z. Xiao, W. Song, Q. Chen, "Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, Issue 6, 2013.

[9] B. Jennings, R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *Springer Journal of Network and Systems Management*, 2015.

[10] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, and S. U. Khan, A. Zomaya, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Springer Journal of Computing*, Vol. 98, Num. 7, 2016.

[11] W. Zhang, H. He, G. Chen and Jilong, "Multiple Virtual Machines Resource Scheduling for Cloud Computing," *International Journal of Applied Mathematics & Information Sciences*, No.5, 2089-2096, 2013.

[12] L. S. Subhash and Dr. K. P. Thooyamani, "Allocation of Resource Dynamically in Cloud Computing Environment Using Virtual Machines," *International Journal of Advancements in Technology* Volume 8. doi:10.4172/0976-4860.1000193, 2017.

[13] W. Tian, Y. Zhao, Y. Zhong, M. Xu, C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for Cloud Datacenters," *IEEE, International Conference on Cloud Computing and Intelligence Systems* DOI: 10.1109/CCIS.2011.6045081, 2011.

[14] A. Jaikar, H. Dada, G. R. Kim, S. Y. Noh, "Priority-based Virtual Machine Load Balancing in a Scientific Federated Cloud," *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014.

[15] L. Lui, Z. Qiu, J. Dong, "A Load Balancing Algorithm for Virtual Machines Scheduling in Cloud Computing," *The 9th International Conference on Modelling, Identification, and Control (ICMIC 2017)*, 2017.

[16] J. Hu, J. Gu, G. Sun, T. Zhao, "A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment," *IEEE, 3rd International Symposium on Parallel Architectures, Algorithms and Programming*, 2010.

[17] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *IEEE, Journal of Parallel and Distributed Computing* Vol. 72, Issue 5, May 2012.

[18] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE, Internet Computing* Volume: 13, Issue: 5, Sept.-Oct. 2009.

[19] E. Kumari and Monika, "Review on Task Scheduling Algorithms in Cloud Computing," *International Journal of Science, Environment and Technology*, vol. 4, no. 2, 2015.