ALGORITHM FOR HETEROGENOUS COMPUTING

Name : DAC Algorithm (Divide And Conquer) [just a suggestion not compulsory]

Divide resembles division of tasks into three queues.
Conquer resembles using of virtual machies as per the requirement based on the divided queues.

- K Naveen Kumar

---

1. Inputs :

- Cloudlets/Tasks with specifications

Cloudlets /Tasks	ID	Number of Instructions	RAM	Size	Arrival Time	Burst Time	DeadLine	Priority
T_1	1	I_1	r_1	S_1	A_1	B_1	DL_1	P_1
T_2	2	I_2	r_2	S_2	A_2	B_2	DL_2	P_2
T_3	3	I_3	r_3	S_3	A_3	B_3	DL_3	P_3
T_4	4	I_4	r_4	S_4	A_4	B_4	DL_4	P_4
T_5	5	I_5	r_5	S_5	A_5	B_5	DL_5	P_5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
T_n	n	I_n	r_n	S_n	A_n	B_n	DL_n	P_n

Where **n** indicates the number of tasks.

- Hosts(Heterogenous) with specifications.

Hosts	ID	MIPS	RAM	Storage	Band Width
H_1	1	MI_1	HR_1	HS_1	BW_1
H_2	2	MI_2	HR_2	HS_2	BW_2
H_3	3	MI_3	HR_3	HS_3	BW_3

H_4	4	MI_4	HR_4	HS_4	BW_4
H_5	5	MI_5	HR_5	HS_5	BW_5
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
H_m	m	MI_n	HR_n	HS_n	Bw_n

Where **m** incidates the number of hosts and **m < n .** The hosts are heterogenous implies
specification(H_1) ≠ specification(H_2) ≠ specification(H_3) ≠ specification(H_4) ≠ specification(H_5) ≠ ....specification(H_m).

- Cost of creating new VM - **β (say Beta).**

2. Outputs :

- ATT (Average TurnAroundTime).
- AWT(Average Waiting Time).
- Performance Evalution
  - percentage of Utilization of Resources
  - Total Number of Migrations
  - Number of Instructions executed per task
- Cost Evaluation

Algorithm :

The algorithm is divided into 3 Rounds.

Round 1 :- Division Of Tasks into HQ, MQ ,LQ.

- **HQ** is HighQueue, **MQ** is MidQueue , **LQ** is LowQueue.

- Firstly, cloudlets **[T_1,T_2,T_3,T_4.....T_n]** are sorted based on priority and stored in an array , let the array be **KeyArray.**
  where
  KeyArray = **[T '$_1$,T '$_2$,T '$_3$,T '$_4$.....T '$_n$]** such that

$$P[T'_i] > P[T'_j]$$

forall i < j
i ≠ j
(i,j) Є [1,n]
P is the Priority. Where on a scale of 1 to 10 the priority of 1 being highest and 10 being lowest.

- Let us consider that the elements of the **KeyArray** to be
  [K_1,K_2,K_3,K_4.....K_n]
  such that

  KeyArray[0] =T '$_1$ = K_1
  KeyArray[0] =T '$_2$ = K_2
  KeyArray[0] =T '$_3$ = K_3
  .
  .
  .
  KeyArray[n] =T '$_n$ = K_n

- Now let us calculate the size of HQ , LQ , MQ . We denote them by **N_1,N_2,N_3.**
  Where **N_1 = Floor { n / 3 } + λ**
  N_2 = Floor { (n – N_1) / 2) }
  N_3 = n - (N_1 + N_2)

  Where **n** is the total number of tasks in KeyArray . **λ** is a positive integer, which help to regulate the number of tasks in HQ, MQ ,LQ respectively.

- Now we have
  KeyArray , N_1,N_2,N_3.
  we now divide the tasks in the KeyArray into HQ,MQ,LQ

  HQ = [K_1,K_2,K_3.....K_{t1}] where t_1=N_1

  MQ = [K_{t1+1},.....K_{t2}] where t_2-t_1=N_2

  LQ = [K_{t2+1},.....K_{t3}] where t_3-t_2=N_3

- Now in the next step, tasks in HQ,MQ,LQ are internally sorted based

on Deadlines.i.e., for example tasks in HQ are sorted in the following manner.

HQ=[K_1,K_2,K_3,.....K_{N1}] is sorted to

[K'_1, K'_2, ...K'_{N1}] where

DL [K'_i] < DL [K'_j]
forall i < j
i ≠ j
(i , j) Є [1 , n]
DL indicates Deadline.

Similarly sort MQ and LQ.

Note : While sorting if deadlines are same for two tasks then sorting is done based on priority i.e., the one among them with highest priority comes first.

- Now calculate Time Quantums for RoundRobin for HQ and MQ .Let us name them as **TQ_1,TQ_2** respectively.
  Where

TQ_1 = Floor { max (Deadlines of Tasks in HQ) / $2^{\alpha 1}$ }

TQ_2 = Floor { max (Deadlines of Tasks in MQ) / $2^{\alpha 2}$ }

α_1 ,α_2 are positive integers .
$\alpha_1 > \alpha_2$

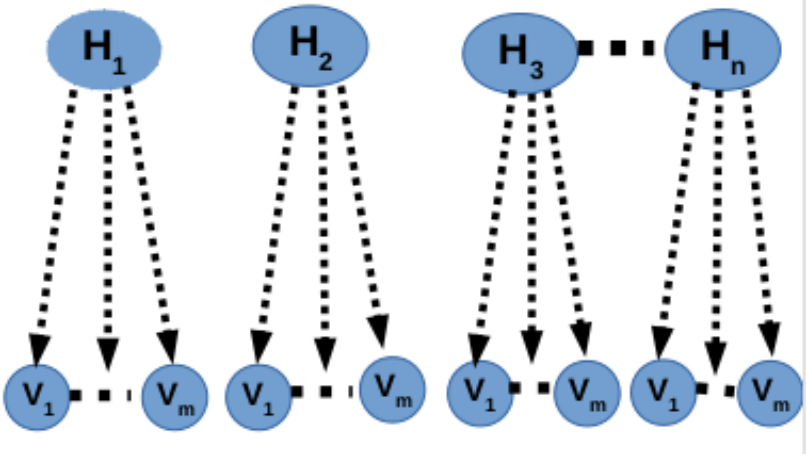
We need to ensure that $TQ_1 < TQ_2$,if it doesnt hold true then increase α_1 by 1 . It is done in the following way

```
while(1):
    if TQ1>TQ2 and TQ1>0:
        ALPHA1=ALPHA1+1
        TQ1=Floor{max(DL)/pow(2,ALPHA1)}
    else:
        break
```

- Round 1 Finishes here. At the end of Round1 we have given tasks divided into three queues, HQ,MQ,LQ and we have calculated Timequantums for roundrobin algorithms to be implemented for HQ,MQ respectively.

Round 2 : Creation of Virtual Machines

Fig1 Scenario of Heterogenous hosts and VM's



- We have ,
  Hosts **[H_1 - H_n]**
  We need to create
  Virtual Machines **[V_1 – V_m]**

- Fig1 shows the resources i.e., the hosts and created Virtual machines allocated to given hosts.

- Dotted arrows indicate that we dont have any information about how many virtual machines need to be created.

- Dotted lines between virtual machines indicate there are **m** number of virtual machines and we dont have any information about **m.**

Stratergies for the creation of Virtual Machines :

- In our algorithm there are three strategies for the creation of virtual machines. There are
  - Average strategy
  - Minimum strategy
  - Default strategy

- **Average strategy :**
  - We start creation of virtual machines based on this strategy and if it fails, we go for next two stratergies.

  - We have to execute **HQ** first , so we create **n VMs** and allocate them to **n** hosts in the following manner. **n** indicates the number of hosts.

    Note: Here hosts as well as vms are heterogenous.

  - In this strategy we basically average the specifications of the tasks in **HQ** and create a virtual machine say **VM1** and we build the other **VMs** based on the **VM1**
    let
    $\mathbf{AN_{HQ}}$ - Average of the number of instructions of tasks in HQ.
    $\mathbf{AR_{HQ}}$ - Average of the RAM of tasks in HQ
    $\mathbf{AS_{HQ}}$ - Average of the size of tasks in HQ

Virtual Machines	**ID**	**RAM**	**Storage**	**MIPS**
VM_1	1	AR_{HQ}	AS_{HQ}	AN_{HQ}
VM_2	2	$AR_{HQ} + HR_2 / 2$	$AS_{HQ} + HS_2 / 2$	$AN_{HQ} + MI_2 / 2$
VM_3	3	$AR_{HQ} + HR_3 / 2$	$AS_{HQ} + HS_3 / 2$	$AN_{HQ} + MI_3 / 2$
VM_4	4	$AR_{HQ} + HR_4 / 2$	$AS_{HQ} + HS_4 / 2$	$AN_{HQ} + MI_4 / 2$
VM_5	5	$AR_{HQ} + HR_5 / 2$	$AS_{HQ} + HS_5 / 2$	$AN_{HQ} + MI_5 / 2$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
VM_n	n	$AR_{HQ} + HR_n / 2$	$AS_{HQ} + HS_n / 2$	$AN_{HQ} + MI_n / 2$

Here VM_1 is allotted to H_1,VM_2 to H_2,.....VM_n to H_n.

Constraints:

- We need to ensure

  $spec(VM_1) \leq spec(H_1)$
  $spec(VM_2) \leq spec(H_2)$
  $spec(VM_3) \leq spec(H_3)$

.
.
.
.
.
spec(VM_n) ≤ spec (H_n)

- If any of this condition fails then we use minimum strategy for that failed VM.

- **Minimum strategy :**
  In this strategy we basically take minimum of the specifications of the tasks in **HQ** and create a virtual machine say **VM1** and we build the other **VMs** based on the **VM1**
  let
  MN_{HQ} - Minimum of the number of instructions of tasks in HQ.
  MR_{HQ} - Minimum of the RAM of tasks in HQ
  MS_{HQ} - Minimum of the size of tasks in HQ

Virtual Machines	**ID**	**RAM**	**Storage**	**MIPS**
VM_1	1	MR_{HQ}	MS_{HQ}	MN_{HQ}
VM_2	2	MR_{HQ} + HR_2 / 2	MS_{HQ} + HS_2 / 2	MN_{HQ} + MI_2 / 2
VM_3	3	MR_{HQ} + HR_3 / 2	MS_{HQ} + HS_3 / 2	MN_{HQ} + MI_3/ 2
VM_4	4	MR_{HQ} + HR_4 / 2	MS_{HQ} + HS_4 / 2	MN_{HQ} + MI_4/ 2
VM_5	5	MR_{HQ} + HR_5 / 2	MS_{HQ} + HS_5 / 2	MN_{HQ} + MI_5/ 2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
VM_n	n	MR_{HQ} + HR_n / 2	MS_{HQ} + HS_n / 2	MN_{HQ} + MI_n / 2

- **Here VM_1 is allotted to H_1,VM_2 to H_2,.....VM_n to H_n.**

- **Constraints:**
- We need to ensure
  spec(VM_1) ≤ spec (H_1)
  spec(VM_2) ≤ spec (H_2)

$spec(VM_3) \leq spec\ (H_3)$

.

.

.

.

.

$spec(VM_n) \leq spec\ (H_n)$

- If any of this condition fails then we use Default strategy for that failed VM.

- **Default Strategy:**

In this strategy we directly take half of the specifications of the Host1 and create a virtual machine say **VM1** and we build the other **VMs** based on the **VM1** adding half of the specifications.

Virtual Machines	**ID**	**RAM**	**Storage**	**MIPS**
VM_1	1	$HR_1/2$	$HS_1/2$	$MI_1/2$
VM_2	2	$HR_1/2 + HR_2/2$	$HS_1/2 + HS_2/2$	$MI_1/2 + MI_2/2$
VM_3	3	$HR_1/2 + HR_3/2$	$HS_1/2 + HS_3/2$	$MI_1/2 + MI_3/2$
VM_4	4	$HR_1/2 + HR_4/2$	$HS_1/2 + HS_4/2$	$MI_1/2 + MI_4/2$
VM_5	5	$HR_1/2 + HR_5/2$	$HS_1/2 + HS_5/2$	$MI_1/2 + MI_5/2$
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
VM_n	n	$HR_1/2 + HR_n/2$	$HS_1/2 + HS_n/2$	$MI_1/2 + MI_n/2$

Here VM_1 is allotted to H_1,VM_2 to H_2,.....VM_n to H_n.

This strategy cannot be failed.

Reasons to choose these strategies :

- **Average strategy** : It ensures maximum utilization of Hosts and all the tasks with specifications less than average can be easily executed with VM's created and the cost of creation of new VM's will be reduced.

- **Minimum strategy** : It comes into picture only when Average strategy fails.Taking minimum ensures full utilization of resources for minimum tasks.

- **Default strategy** : It comes into picture only when both the above strategies fail . This strategy would never fail as we directly take half of the resource from the given one without any concern with the given tasks.

  Note :

- While creation , at first all VM's will follow average strategy. Then we go for constrant check . Only the failed VM's will follow the minimum strategy i.e, they are created such that

  $spec(VM_k) = spec(VM_1) + spec(H_K) / 2$.

  where **$spec(VM_1)$ = min { (spec(tasks of HQ) }, spec : specifications.**

- **Maximum strategy is not taken into account after average strategy failed. The reason is , failure of average strategy implies failure of maximum strategy because maximum is always more than average. Hence there is no meaning in taking maximum strategy and again checking for constraints.**

- So, at the end of Round2 we have VM's created and allocated to Hosts.

Round 3 : Allocation of Tasks to VMs

- Firstly HQ is executed using RoundRobin algorithm with the time quantum of TQ_1 **(**calculated in Round1) in the following way.

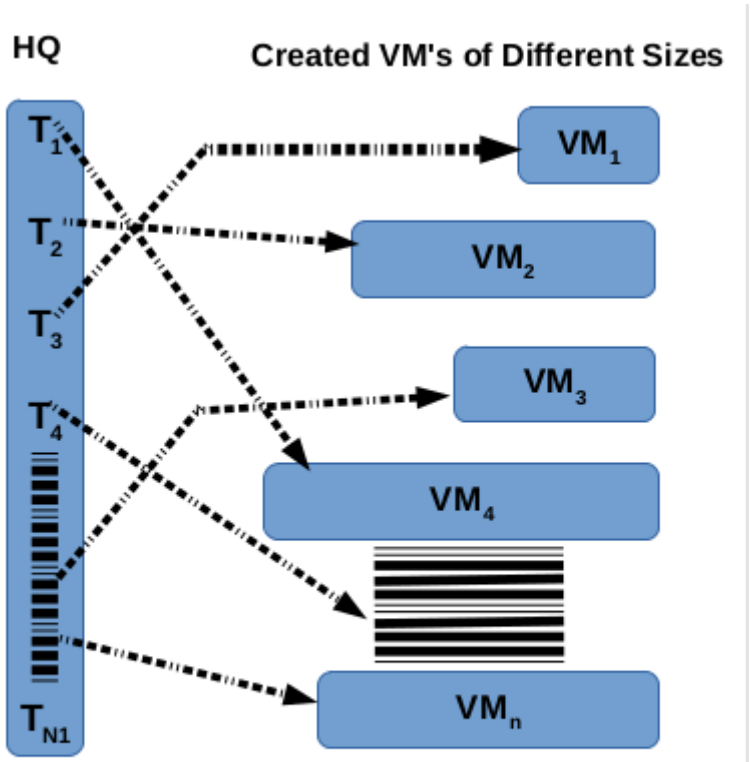


FIG2 Allocation of tasks to VM's .

- Fig2 indicates the tasks in HQ [T_1 - T_{N1}] allocated to heterogenous virtual machines. Different sized virtual machines indicate that they are heterogenos. Dotted arrows indicate that tasks are allocated based on best fit and we have no information regarding which task is allocated to which virtual machine. Dotted line inside HQ indicated that there are N_1 tasks . Similarly dotted lines in the virtual machines indicate that there are n virtual machines.

- Tasks are alloted to virtual machines in the best way in the following way

  $| T_K - VM_H | < \mu$

  where μ is threshold. We need to set it.

- After allocation of tasks to virtual machines ,they are executed till the Timequantum TQ_1 and if the DL of particular task is less than TQ_1 then task is executed till the DL and switching of tasks takes place.

- Here migration will also take place among the tasks allocted in the initial time quantum i.e., for example suppose T1 is allotted to VM3 , T2 – VM6, T3 – VM9 and so on, after first timequantum switching when the turn for the firstly allotted tasks arrives instead of going again into the same virtual machine they again look for available best fit Virtual Machine. Hence migration of tasks takes place.

- This ensures less waiting of tasks and ensures maximum utilization of VM's created and avoids further more creation of virtual machines there by reducing the cost.

- Similarly as soon as tasks in HQ complete execution . Tasks in MQ start execution in roundrobin fashion with time quantum TQ2 (calculated in round1). Similar procedure is followed again . Finally LQ tasks are executed in FCFS fashion.

Conditions / Constraints of Round3 :

- **if (capacity of T_i > (capacity (created VM's)))**
  then
  if (capacity of T_i > (capacity (given Hosts)))
  then
  Throw an exception to the user

  else
  create new VM with specification equal to T_i
  and allocate it to largest availabe Host.

- After completion of tasks in HQ the extra VM's created are destroyed.
  Reason :
  - **To create space for new virtual machines.**
  - **Since they are created specially to meet the specifications some tasks.**
  - **They dont follow any strategy.**

- Killing a Virtual machine among the intially created virtual machies which were created based on strategy is also done based on the following way

  If VM_k remains idle for two time quanta (TQ) i.e., unused by any task then kill it and create new virtual machine with specifications as

  $$\mathbf{spec\ (VM'_k) = spec\ (VM_k) + (H_k / 2^{\theta})}$$

  where $\theta = \{1,2,3,4,5.....\}$ spec : specifications
  and check whether these specifications are less than or equal to the host specifications. If not then go for next θ value.

Result Analysis :

- In every execution Turn around time and waiting time , % of execution is calculated based on the formulae.

  TurnaroundTime = Completion time – Arrival time.

  Waiting Time = TurnaroundTime – Burst time.

  % of execution =
  (time of execution of task before deadline / total burst time) * total number of instructions for that task.

- Average is done to find ATT – Average turn around time, AWT – Average waiting time. Total number of instructions executed.

- % of utilization will be calculated based on the number of number of VM's created and used at the end of HQ ,MQ ,LQ respectively.

- Fair comparision among different other algorithms like SJF and FCFS is done by keeping the creation strategy i.e., round 2 as same and new virtual machine creation strategy as same and varying round1 and round3.

Cost Evaluation :

- Given **β** as cost of creating a single VM. As the cases proceed + **β** will be added everytime a new virtual machine is created.

- Cost Matrix (CM) will be

Tasks

Strategy	**1**	**2**	**3**	**4**	**5**	**.**	**.**	**.**	**n**
Average									
Minimum									
Default									

CM (i , j) = cost for j tasks for i^{th} strategy.
CM (1 ,) = cost for Average strategy.

CM (2 ,) = cost for Minimum strategy.
CM (3 ,) = cost for Default strategy.

- Overall cost includes :

$$CPE = \beta + S_{UT} + D_{RT}$$

CPE : Cost Performance Efficient.

β : given cost of creation of new virtual machines.

S_{UT} : sum of usage times of VM's (how much time VM is in use , i.e., resource in utilization)

D_{RT} : Difference between response time (i.e., time between one task arrival to another into the VM i.e., delay)

Reasons to calculate in this fashion :

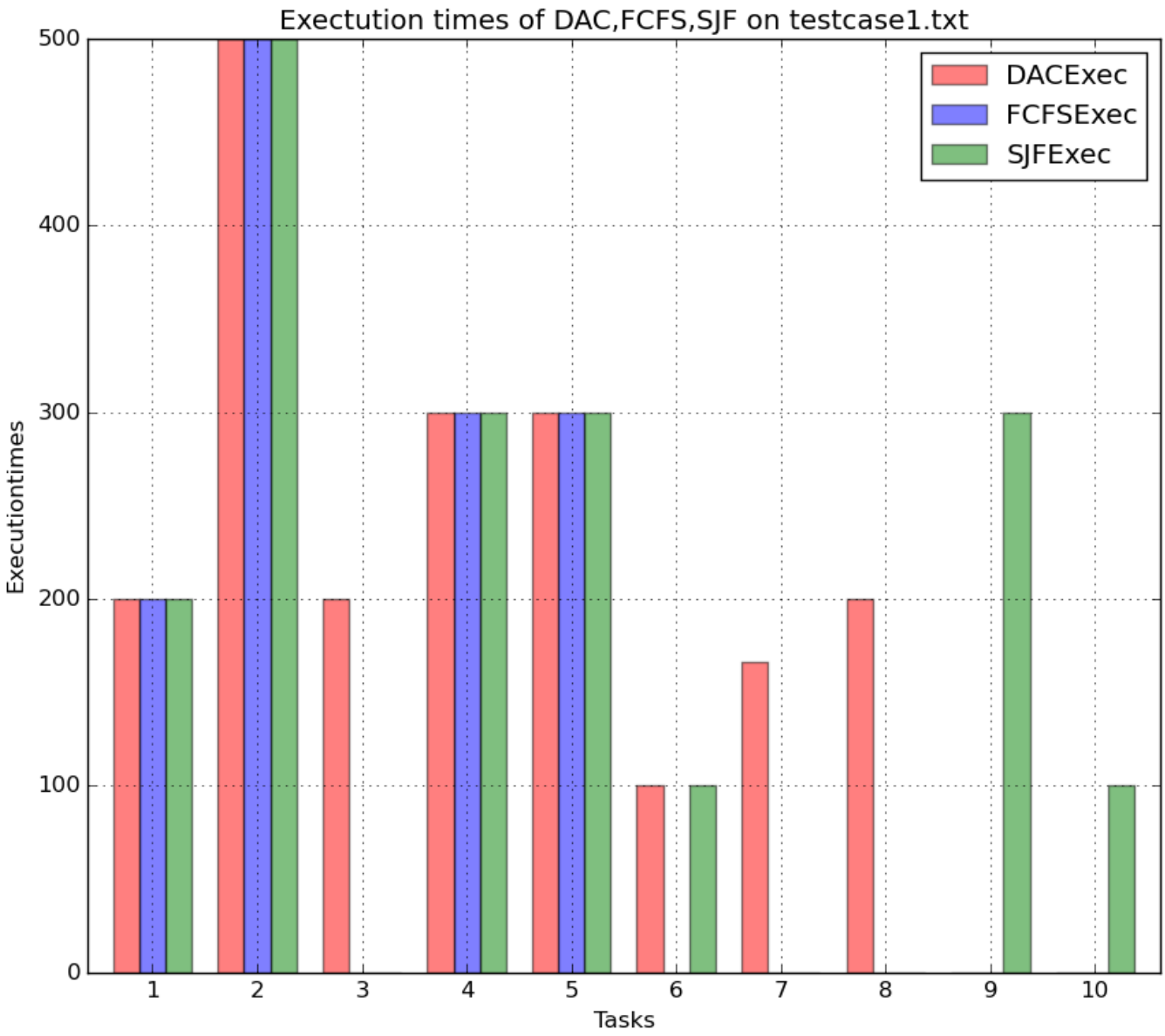
- **Since cloud is cost associative ,every utilization must be taken into account as long as the resource is utilized i.e., from the time of creation to the time of killing of VM.**

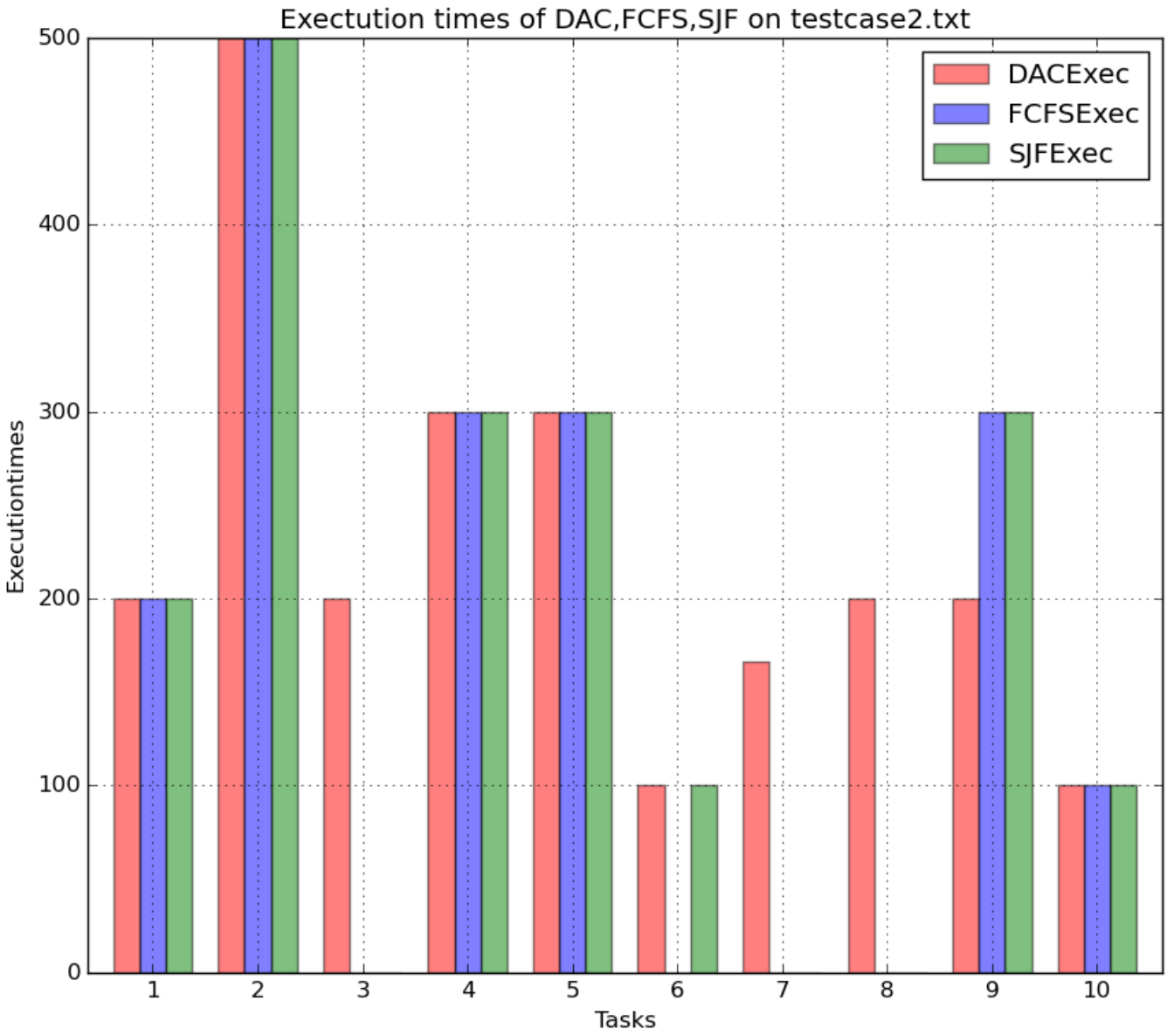
- **For example electricity bill includes consumption of electricity by every device and the whole bill is calculated accordingly.**

1. **This algorithm ensures, atleast partial execution of tasks having very less deadlines.**
2. **Ensures maximum utilization of resources.**

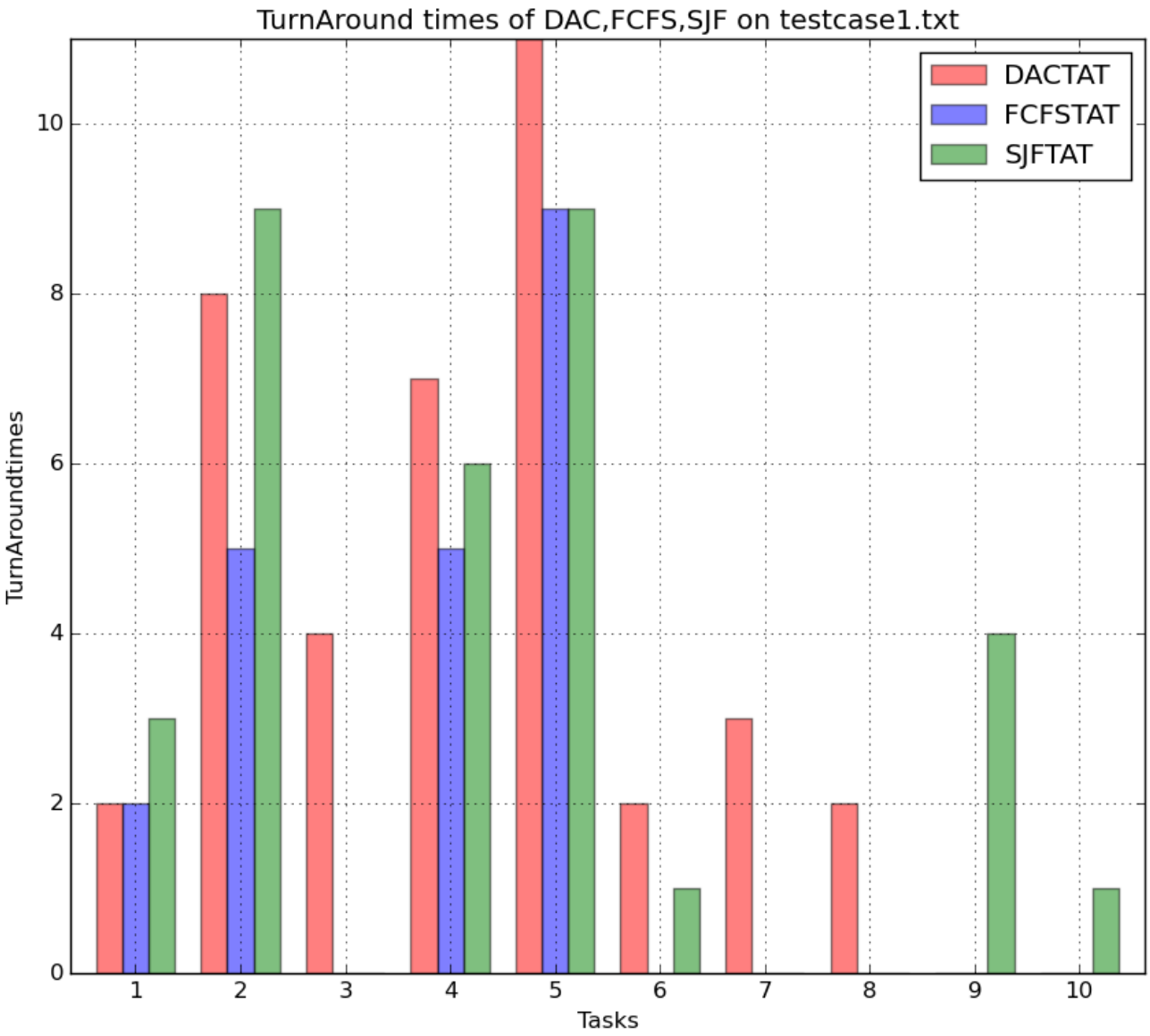
Outputs and Observations :

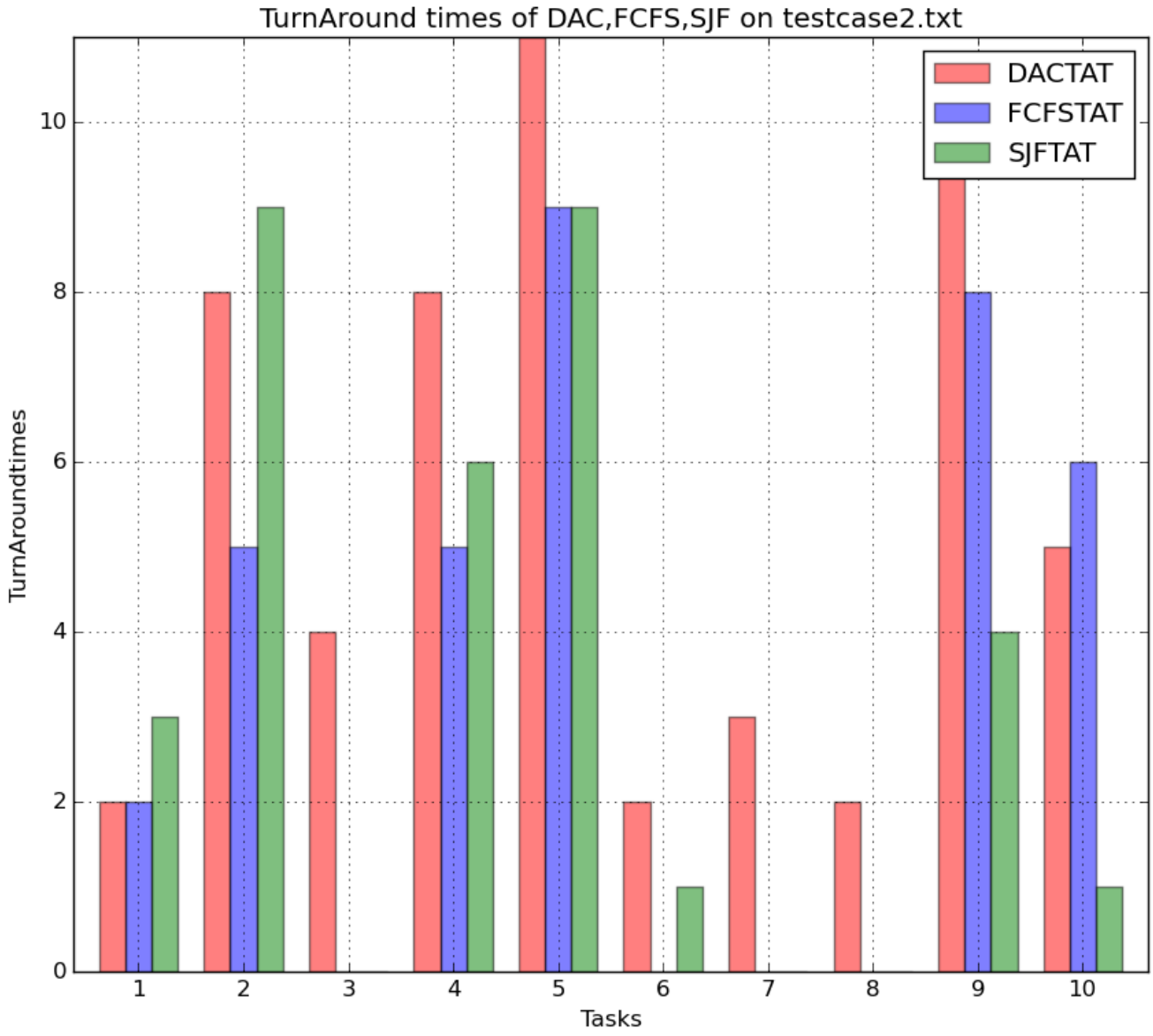
Execution Time



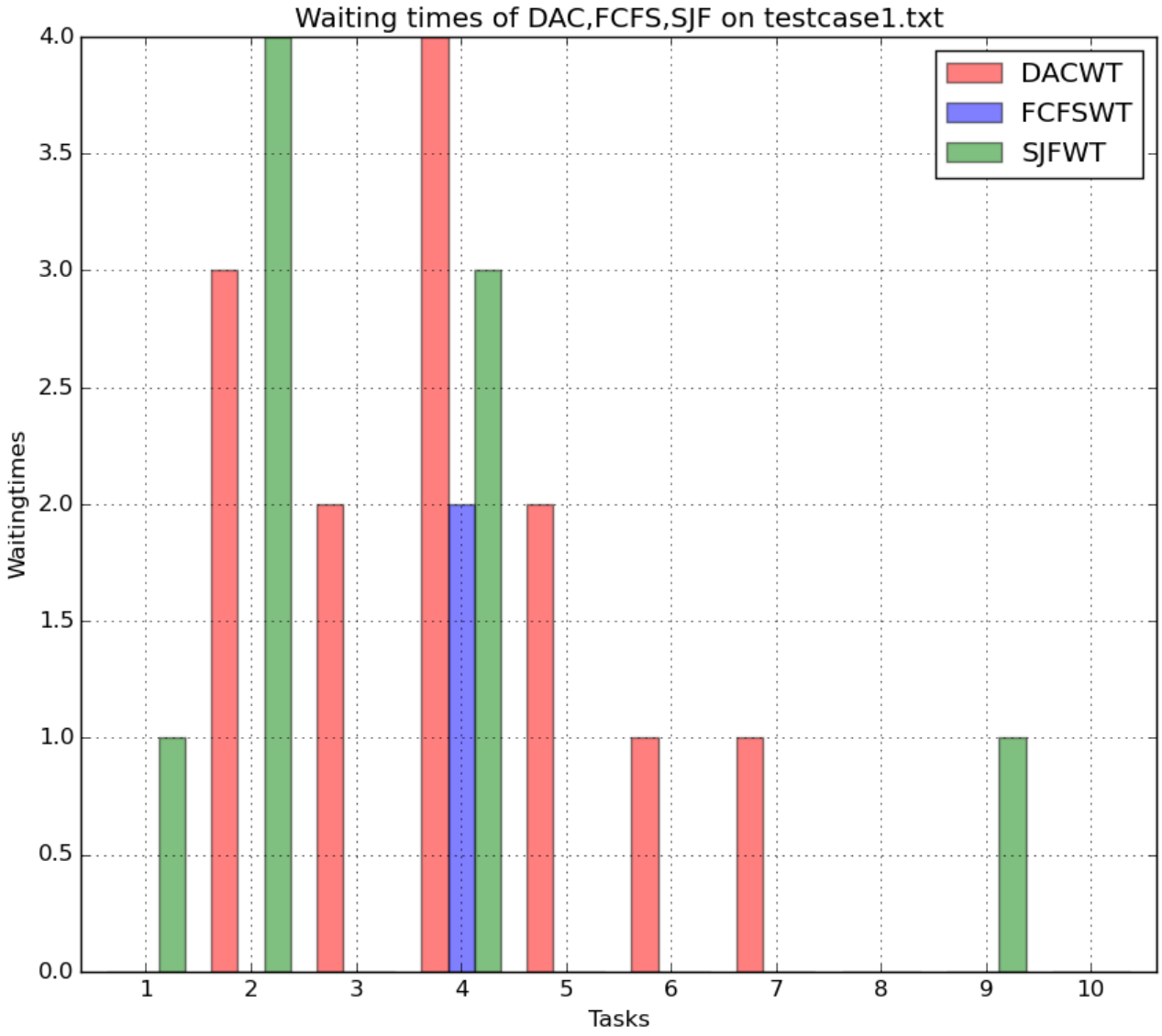

Exectution times of DAC,FCFS,SJF on testcase2.txt
DACExec
FCFSExec
SJFExec
Executiontimes
Tasks
0
100
200
300
400
500
1
2
3
4
5
6
7
8
9
10

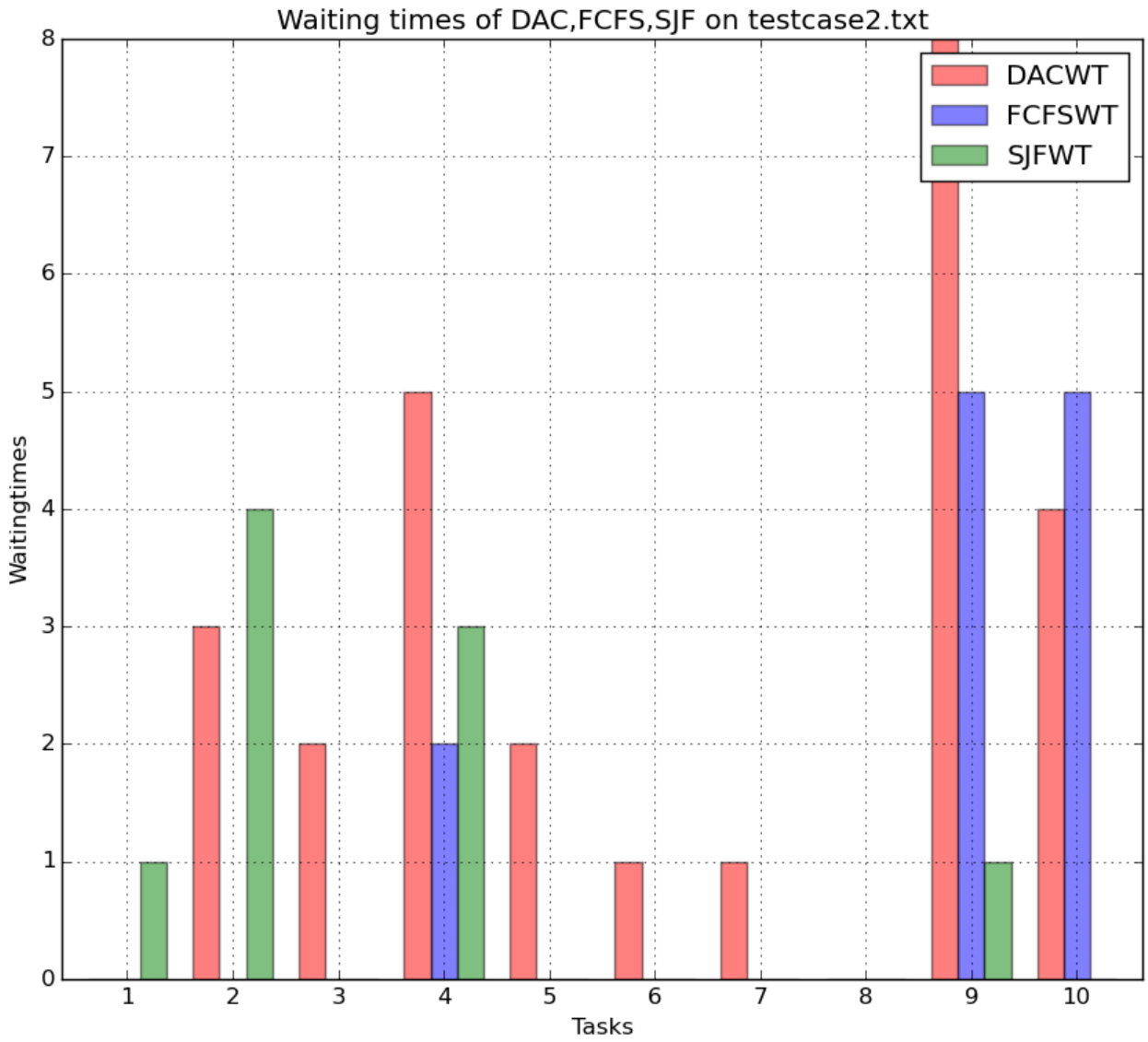
TurnAround Time




TurnAround times of DAC,FCFS,SJF on testcase2.txt
DACTAT
FCFSTAT
SJFTAT
TurnAroundtimes
Tasks
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
10

Waiting Time




Waiting times of DAC,FCFS,SJF on testcase2.txt
DACWT
FCFSWT
SJFWT
Waitingtimes
Tasks
0
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
9
10

Observations :

- It is observed that, in the above two test cases., DAC allows
  - Minimum partial execution(provided for a given deadline and high priority)
  - with a minimum waiting and turn around times

- When it comes to Number of Instructions executed , FCFS and SJF never exceed DAC execution and when they are 0, DAC allows partial execution.

- In comparision with the standard FCFS and SJF algorithms, performance of DAC is better in terms of Execution.