**FIRST COME FIRST SERVE SCHEDULING ALGORITHM**

This scheduling algorithm assigns jobs ----🡪oldest first i.e. the file that comes first is processed and stored first.

In general, there are two types of jobs, CPU-bound and I/O-bound, which require different resources but run simultaneously in the same cluster. The default job scheduling policy of Hadoop is ﬁrst-come-ﬁrst-served and therefore, may cause unbalanced resource utilization. Considering various job workloads, numerous job allocation schedulers were proposed in the literature. However, those schedulers encountered the data locality problem or unreasonable job execution performance. This study proposes a job scheduler based on a dynamic grouping integrated neighbouring search strategy, which can balance the resource utilization and improve the performance and data locality in heterogeneous computing environments.

**Code in c: {FCFS implemented}**

#include<iostream>

using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

for (int i = 1; i < n ; i++ )

wt[i] = bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

**PROPOSED SCHEDULING ALGORITHM**

Algorithm 1: JOB CLASSIFICATION (Heartbeat) {DETERMINATION OF TYPE OF JOB}

1 get TaskTracerQueues information from Heartbeat:

2 Initialize LocalityBeneﬁtTable: = null;

3 for each task in the Task Tracker do

4 if the task has been completed by the Task Tracker then

5 obtain the task information from the Task Tracker;

6 compute throughputs: = n∗MTCT (MAP TASK COMPLETED TIME) where n is total number of map tasks;

7 if the task belongs to a job J that has not been classiﬁed then

8 if throughput < DIOR (Disk average input output rate) then

9 set J as a CPU-bound job;

10 move J to the CPU Queue;

11 else

12 set J as a IO-bound job;

13 move J to IO Queue;

14 if the task belongs to a CPU-bound job then

15 record the execution time of the task on TaskTrackerCPUCapability;

16 else

17 record the execution time of the task on TaskTrackerIOCapability;

18 record the execution time of the task on LocalityBeneﬁtTable.

Algorithm 2 :-SLOT CREATION IN DATA NODES

1.Assign Cpu And Io Slots From The Information In The TASK TRACKER IO AND CPU TABLE;

Assingning jobs to the task tracker through this algorithm will solve the problem of data locality and unbalanced resource utilization.