Job Scheduling using FCFC and Priority Queue in System

Monika¹, Neelam²
¹M. Tech Scholar, CSE Dept, MMU, Sadopur, India
²Assitant Professor, CSE Dept, MMU, Sadopur, India
¹mlakra89@gmail.com

Abstract: nowadays job scheduling has become a popular platform for scientific applications. Job scheduling intends to share a large number of resources such as equipment’s for storage and computation, and information and knowledge for scientific researches. Job scheduling algorithm is one of the most challenging theoretical issues in the computing area. How to use computing resources efficiently and increase user satisfaction with jobs scheduling system is one of the computing service provider’s important goals. Some intensive researches have been done in the area of job scheduling of computing. In this research work we have proposed Hybrid Algorithm in computing. In order to prove our opinions we will process this work as the following steps. First of all, we will declare some task which we want to execute. Then, according to the tasks selection, we will select the appropriate branch of the function and compute the justice evaluation. A hybrid algorithm is combination of FCFS and Priority concept. This will also reflects the hybrid algorithm to select local optimum. Compare to other methods like FCFS, it can decrease the completion time of submitted tasks and increases the user satisfaction.

Keywords: Job Scheduling, Operating System, FCFS, Priority Concept.

I. INTRODUCTION

Job scheduling is a term in which we take some jobs and send them to the scheduler to execute them. Now the question is how efficiently a job can be executed so that least amount of energy gets consumed. New parallel computing systems, such as the SUN Microsystems E10000, the SRC-6, and the SGI Origin 2000, provide a pool of homogeneous processors, a large shared memory, customizable I/O connectivity, and expandable primary and secondary disk storage support [1].

Each resource in these system architectures may be scaled independently based on cost and user need. A site which typically runs CPU intensive jobs may opt for a configuration which is fully populated with CPUs but has a reduced memory to keep the overall system cost low. Alternatively, if the expected job mix contains a large percentage of I/O and memory intensive jobs, a large memory configuration may be purchased with high I/O connectivity to network or storage devices. Finally, a mixed job set may be best serviced by a balanced system configuration. Therefore, given an expected job mix, a "shared-everything" parallel system can be configured with the minimal set of resources needed to achieve the desired performance [2]. The question, then, is how to schedule jobs from the actual job stream onto a given machine to achieve the expected performance. This is the K-resource scheduling problem.

Consider extending the FCFS-based schemes to account for multiple (K) resources in a particular physical system configuration. The pure FCFS job allocation scheme would pack jobs from the job queue into the system, in order of their arrival, until some system resource (CPUs, memory, disk space, etc.) was exhausted. In this case, the job allocation scheme is blocked from scheduling further jobs until sufficient resources become available for this large job. This potentially results in large fragments of resources being under-utilized [3]. The FCFS with backfill probabilistically performs better by skipping over jobs which block while waiting for large percentages of a single resource and finding smaller jobs which can make use of the remaining resources. Still, a single resource becomes exhausted while others remain under-utilized.

II. WHERE SCHEDULING FIT IN PROCESS

![Diagram of scheduling process](Image)
III. GENERAL GOALS

1. Fairness: Fairness is important under all circumstances. A scheduler makes sure that each process gets its fair share of the CPU and no process can suffer indefinite postponement. Note that giving equivalent or equal time is not fair. Think of safety control and payroll at a nuclear plant.

2. Policy Enforcement: The scheduler has to make sure that system's policy is enforced. For example, if the local policy is safety then the safety control processes must be able to run whenever they want to, even if it means delay in payroll processes.

3. Efficiency: scheduler should keep the system (or in particular CPU) busy cent precent of the time when possible [4]. If the CPU and all the Input/output devices can be kept running all the time, more work gets done per second than if some components are idle.

4. Response Time: A scheduler should minimize the response time for interactive user.

5. Turnaround: A scheduler should minimize the time batch users must wait for an output.

6. Throughput: A scheduler should maximize the number of jobs processed per unit time.

IV. SCHEDULING ALGORITHMS

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU.

1. FCFS (First Come First Serve)

FCFS stands for “First Come First Serve”. In this algorithm the first data which reaches to the queue first gets executed first. This algorithm is time consuming and does not perform quite efficiently when there is a case of priority in the segmentation. Other names of this algorithm are:

- First-In-First-Out (FIFO)
- Run-to-Completion
- Run-Until-Done

Perhaps, First-Come-First-Served algorithm is the simplest scheduling algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a non-preemptive discipline, once a process has a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait.

FCFS is more predictable than most of other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand. One of the major drawbacks of this scheme is that the average time is often quite long [5]. The First-Come-First-Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

2. Priority Scheduling Algorithm

The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm. The basic idea is straightforward: each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa. Priority can be defined either internally or externally. Internally defined priorities use some measurable quantities or qualities to compute priority of a process.

Priority scheduling can be either preemptive or non-preemptive

- A preemptive priority algorithm will preemptive the CPU if the priority of the newly arrival process is higher than the priority of the currently running process.

- A non-preemptive priority algorithm will simply put the new process at the head of the ready queue.

A major problem with priority scheduling is indefinite blocking or starvation [6]. A solution to the problem of indefinite blockage of the low-priority process is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

V. RESEARCH METHODOLOGY

Step 1- We load data on Microsoft Excel Sheet that works backend and accessed by our project.

Step 2- Use frontend .net accesses that data for implementation of algorithm.

Step 3- We apply concept of FCFS on data known as increasing algorithm and get results.

Step 4- For better performance of new algorithm we use FCFS concept with priority scheduling.
Step 5- First task execute using concept of increasing method and then concept of priority applied.

Step 5- All task exclude first task will implement on basis of priority queue.

Step 6- FCFS and Priority implemented and got results.

Step 7- The result should be better than existing FCFS algorithm.

VI. FLOWCHART OF PROPOSED WORK

VII. RESULTS

We implement our proposed algorithm in MATLAB tool. The following figures shows those tasks which we are going to execute.

Figure 1: Shows window where task will be selected and priority assigned

Figure 2: Graph represents execution time and waiting time for each task in FCFS.

Figure 3: Graph represents execution time and waiting time for each task in Hybrid Approach.

VIII. CONCLUSION

It is expected that the time for task execution will be reduced if we will be implementing the FCFS along with the priority queue concept. Performance of systems is improved with the combination of FCFS and priority queue. In general configuration of system is fixed but in this project we create three systems according our requirement and applied task on these for execution.
FCFS and priority concept is failure to achieve best performance if implemented separately. But a combined approach will make the execution fast and improve performance.

In future if some concept could be applied over the priority as there is no such rule that if a task is of higher priority but a more execution time, it should proceed last can make the system go faster. The main drawback arise in this project is that we have create limited system so that add new system will affect performance of all systems. We need to define priority at time of task selection but in future it can automatically priority allotment on basis of available system configurations. In future we can improve performance by applying priority queue concept on systems on basis of their configuration.

IX. REFERENCES


