Scheduling: Proportional Share

Chapter 9

Previously in CS212…

- We asked the question, “Can we find a compromise for good turnaround time and response time with little (or no) information about the lifetime/compute requirements of a process in advance (*a priori*)?”

- Multi-Level Feedback Queue scheduler compromises between these metrics using multiple queues of jobs with different priorities with an “aging system” to slowly shift batch style jobs to a lower priority and priority boosting to mitigate starvation

- But what if we wanted to emphasize fairness…

The Proportional Share Scheduler

- Also referred to as a **fair-share** scheduler

- Focuses on trying to ensure that each job obtains a certain percentage of CPU time

- Approaches:
  - Lottery Scheduling
  - Stride Scheduling
  - Completely Fair Scheduler (CFS)

Lottery Scheduling

- ”Tickets” are distributed to processes to indicate a share of a resource
  - In our case, CPU time, but could be used for other resources as well
- Winning ticket numbers are chosen randomly by the scheduler
- Processes with more tickets are more likely to ”win” and receive CPU time
  - Effective light-weight approach that is **probabilistically** correct

Lottery Scheduling Example

Job Name	Ticket #s
A	0 - 9
B	10 - 29
C	30 - 99

- What is the likely-hood of running each job:
  - A = 10%
  - B = 20%
  - C = 70%

???

- Scheduler picks: 99, 16, 80, 60, 13, 45, 6, 56, 76, 82, 40, 5, 27, 88, 7
  - C, B, C, C, B, C, A, C, C, C, C, A, B, B, A

- What are the observed scheduling results?
  - A = ~20%
  - B = ~27%
  - C = ~53%

Ticket Mechanism – Ticket Currency

- The scheduler provides a set number of tickets to each job
  - This represents a global currency
- However, a job can allocate tickets to its tasks and in arbitrary numbers
- The scheduler will deal with scaling ticket distribution to the number of tickets provided to that job in the global currency
  - E.g., Job A has two tasks and 60 tickets globally, it can provide 50 and 100 tickets to each of its tasks, but the scheduler scales this to be 20 and 40 global tickets in global currency for the tasks respectively

Ticket Mechanism – Ticket Transfer

- Sometimes a job does not need all the tickets it is provided
- Ticket transfer allows for a job to temporarily donate its tickets to another job
- Can be useful when a job relies on the results another to help maximize resource allocation to get the result faster
- Tickets are **returned to the loaner** after the job is complete

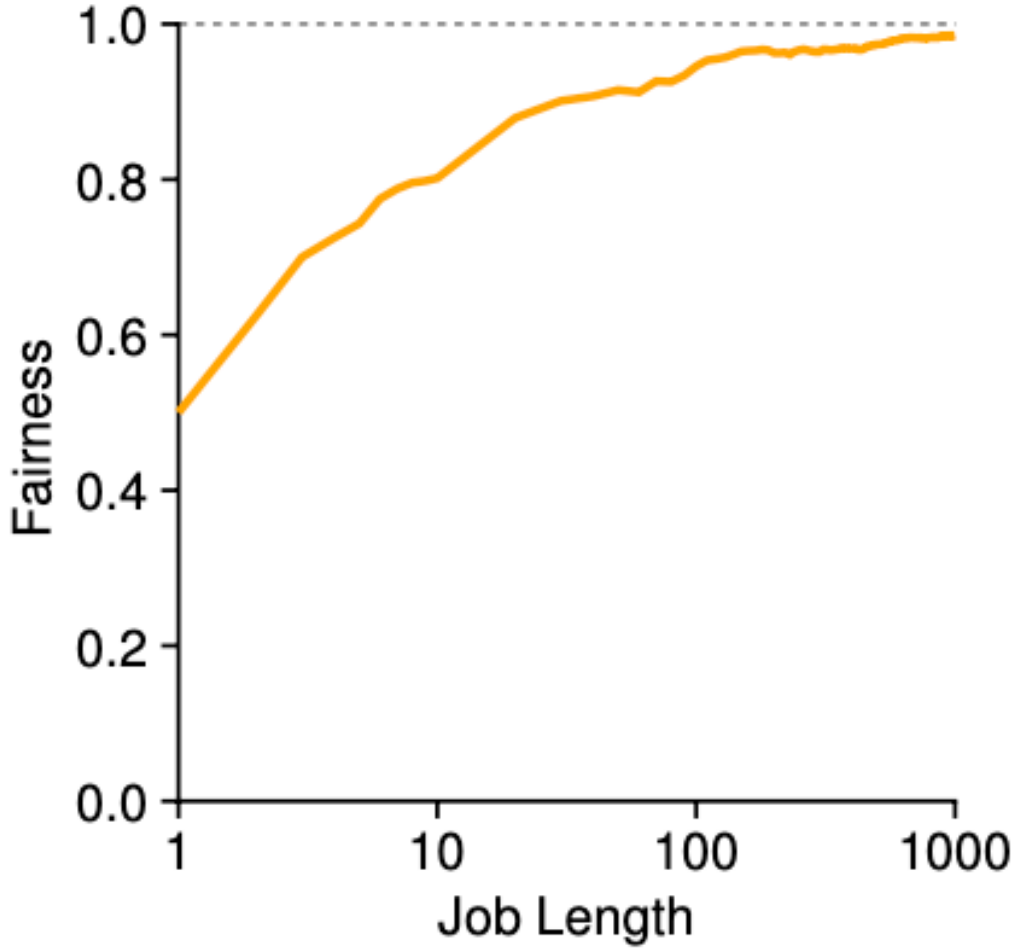
Ticket Mechanism – Ticket Inflation

- A process can temporarily increase or decrease the number of tickets it has (global currency)
- Doesn’t require communication between processes like ticket transfer
- Only really works in a cooperative setting as a greedy process could starve others

Lottery Scheduling Issues

- Requires many time slices before ideal ”fairness” is reached

- How should we assign tickets?



Stride Scheduling

- Attempt to reach a more optimal fairness outcome over shorter time slices by limiting randomness to achieve correct time slice proportions
- While we still assign tickets, we calculate a **stride** for each job by taking is number of tickets and dividing it by a very large number

- As the processes run, we add their stride value to a counter associated with the process (**call the pass value**)

- The scheduler always selects the job with the lowest pass value to run
  - If there is a tie, one may be chosen randomly

Stride Scheduling Example

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

Completely Fair Scheduler (CFS)

- Keeps track of the amount of time a process has run on the CPU with **virtual time** (vruntime)
- Picks the process with the lowest vruntime to run next
- Uses two control parameters:
  - sched_latency – how long should a process run before considering a switch
    - This value is divided by the number of jobs to determine slice time per job
  - min_granularity – the smallest possible time slice for a process
- Utilizes a timer interrupt to frequently wake up and see if a switch is necessary.

CFS Weights

- CFS supports priority scaling as well via a **nice** level
- Nice values range from -20 to +19 where positive values imply lower priority
- A table of constants represents the weight to be applied to the sched_latency (default is 1024)
  - Weight proportion is calculated as the current job weight over the sum of weights for all job

CFS and Weights Example

- Jobs A and B are in the system
  - Assume a sched_latency of 48ms
- Job A is given a priority of -3 (1991) and B a priority of 0 (1024)
- What are the time slices:
  - Time Slice for job A = (1991/ (1991+1024)) * 48 = 32ms
  - Time Slice for job B = (1024 / (1991+1024)) * 48 = 16ms

Process Storage and Selection

- CFS does not use a list to store process in need of scheduling
  - Why?

- CFS uses a **red-black tree** which is like a binary tree, but auto-balances itself to ensure optimal performance
  - Processes are order by vruntime

- The red-black tree only contains running or ready (runnable) processes
  - If a process was blocked, CFS will set its vruntime to that of the minimum value in the tree when it becomes ready again