CS 425 / ECE 428
Distributed Systems
Fall 2026

Aishwarya Ganesan & Ram Kesavan

Lecture 23B: Scheduling

Ack: Indranil Gupta

All slides © IG

Announcements

- HW4 released, due 12/3, Thursday!
- MP4 will be released today
  - Due 12/6, Sunday 11:59pm
  - Demos 12/7, Monday
- HW4 & MP4 due after Thanksgiving/Fall Break, so please start soon!
  - They're **both due the same week**!
- HW3 grades released (regrade window until Nov 16, 11:59pm)

Job vs Task

In distributed systems frameworks like Hadoop or Storm, what is the difference between a Job and a Task?

Responses hidden

A job is an fine-grained unit of execution on a single CPU core, while a task represents the entire user application from start to finish.

A job and task are synonymous terms used interchangeably.

A job is triggered by the user starting an application, and the job is broken down into multiple tasks, some of which may execute in parallel.

A task is a collection of multiple jobs that run sequentially across a cluster.

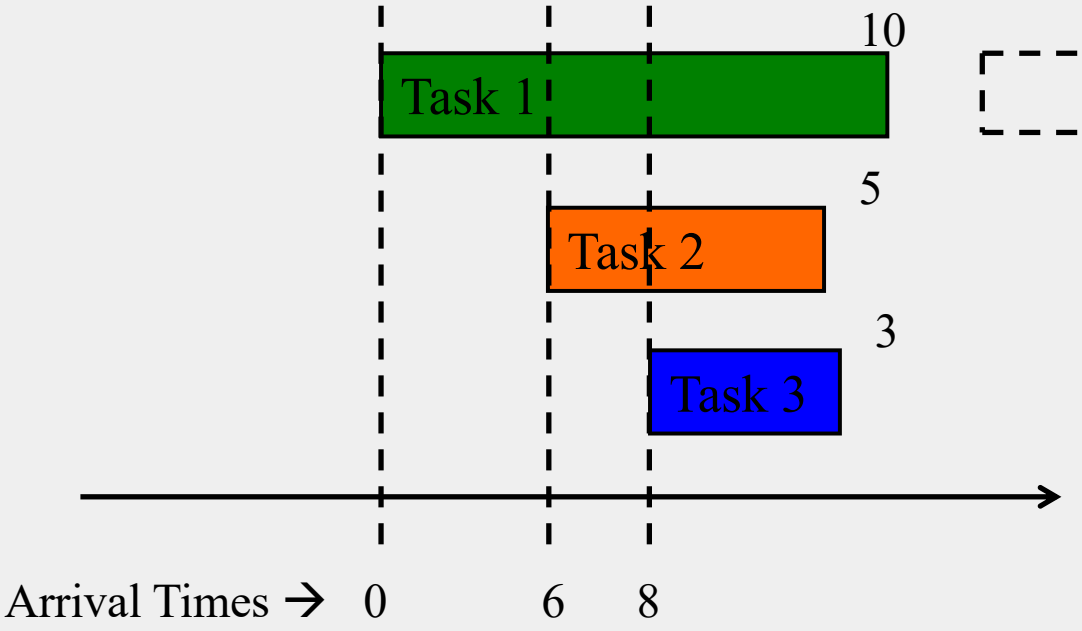
Why Scheduling?



Which metrics are important?
Cloud user vs. provider

- Multiple jobs to schedule in a distributed system
  - Each comprising multiple tasks
- Shared resources that these jobs/tasks require
  - CPU
  - Memory
  - (Less contentious) disk, network, flash
- Scheduling goals
  1. Good throughput or response time for tasks (or jobs)
  2. High utilization of resources

Single Processor Scheduling

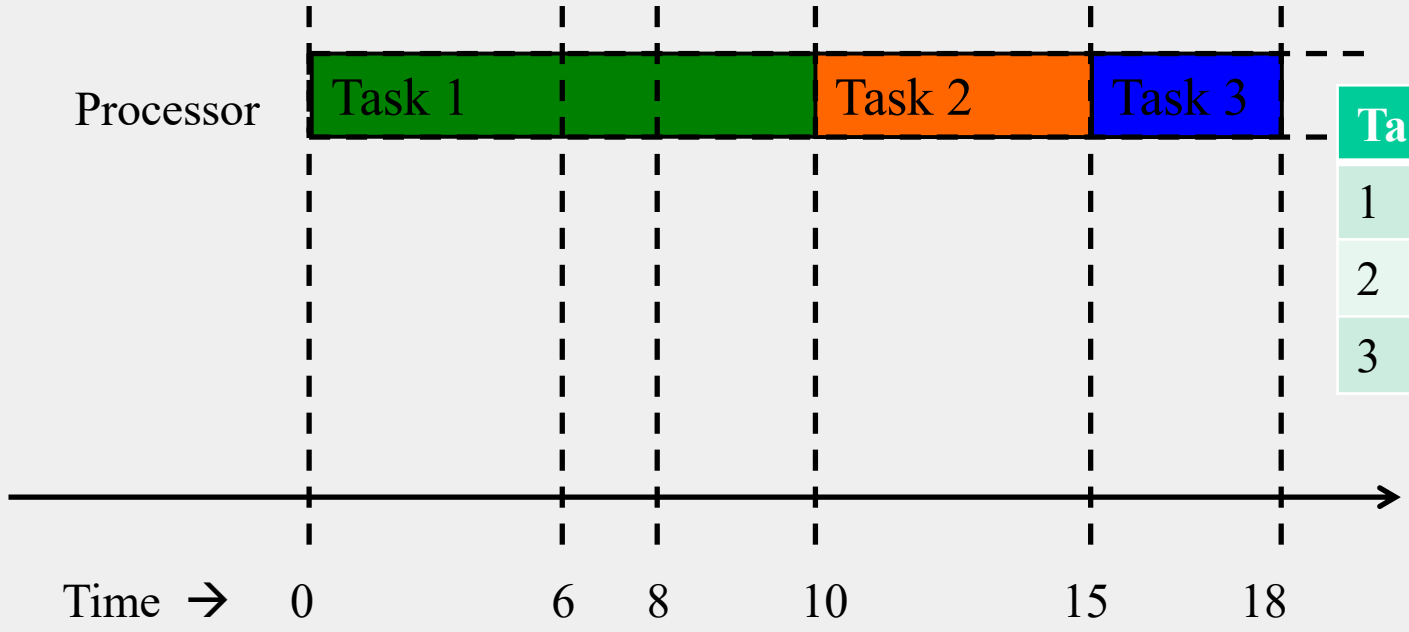


Which tasks run when?

Processor

Task	Length	Arrival
1	10	0
2	5	6
3	3	8

FIFO Scheduling (First-In First-Out)/FCFS



Task	Length	Arrival
1	10	0
2	5	6
3	3	8

- *Maintain tasks in a queue in order of arrival*
- *When processor free, dequeue head and schedule it*

FIFO/FCFS Performance



What are some other disadvantages of FIFO? Advantages?

- Average completion time may be high
- For our example from the previous slides
  - Average completion time of FIFO/FCFS =

    (Task 1 + Task 2 + Task 3)/3

    = (10 + 9 + 10)/3 = 29/3

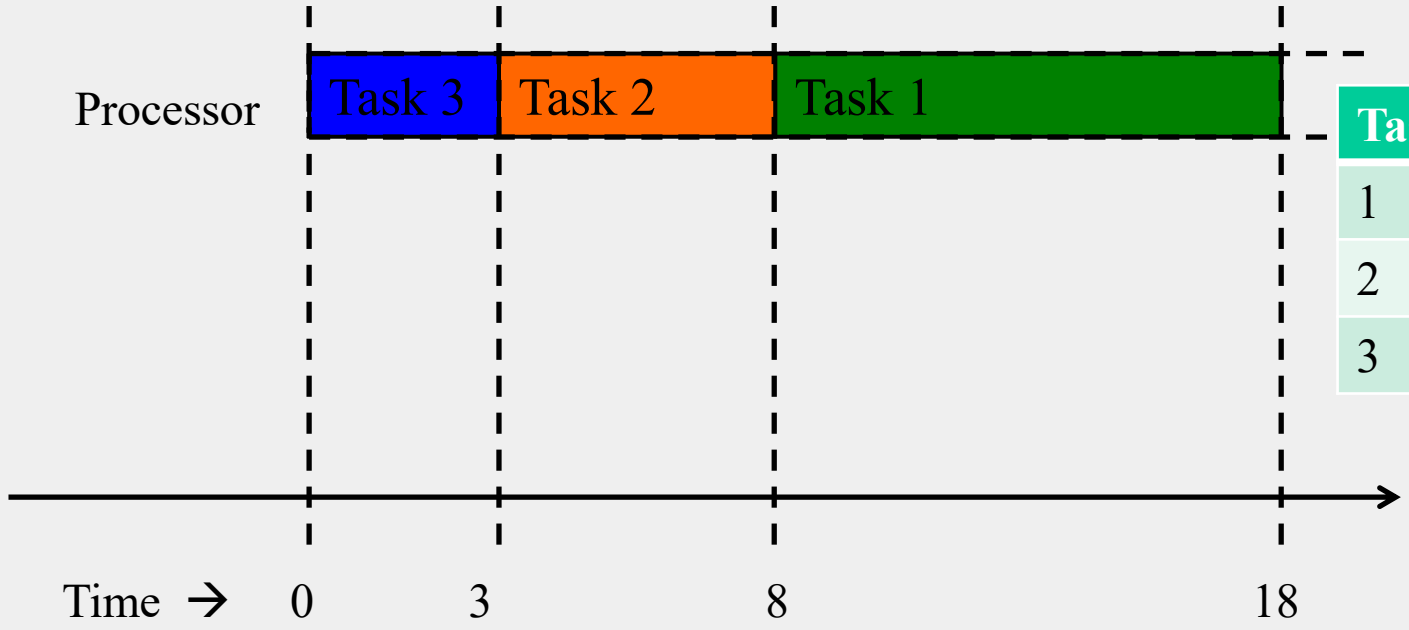
    = 9.66

    But, what if Task2 and Task3 arrived immediately after Task1?

    = (10 + 15 + 18)/3 = 43/3

    = 14.33

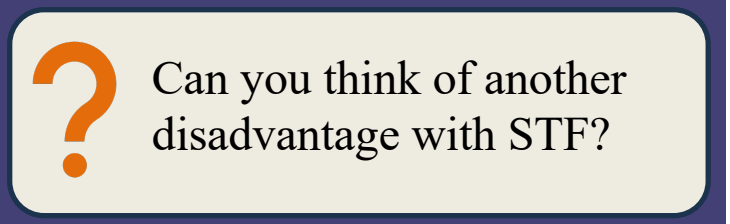
STF Scheduling (Shortest Task First)



Task	Length	Arrival
1	10	0
2	5	0
3	3	0

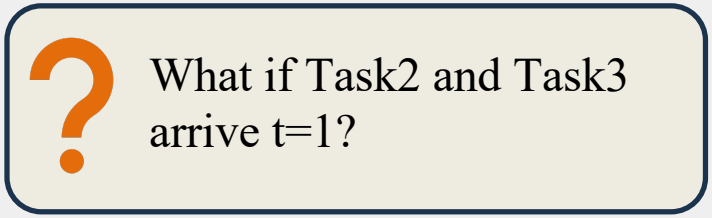
- *Maintain all tasks in a queue, in increasing order of running time*
- *When processor free, dequeue head and schedule*

STF Is Optimal!



- Average completion of STF is the shortest among *all* scheduling approaches!
- For our example on previous slides,
  - Average completion time of STF = (Task 1 + Task 2 + Task 3)/3
  - = (18+8+3)/3
  - = 29/3
  - = 9.66

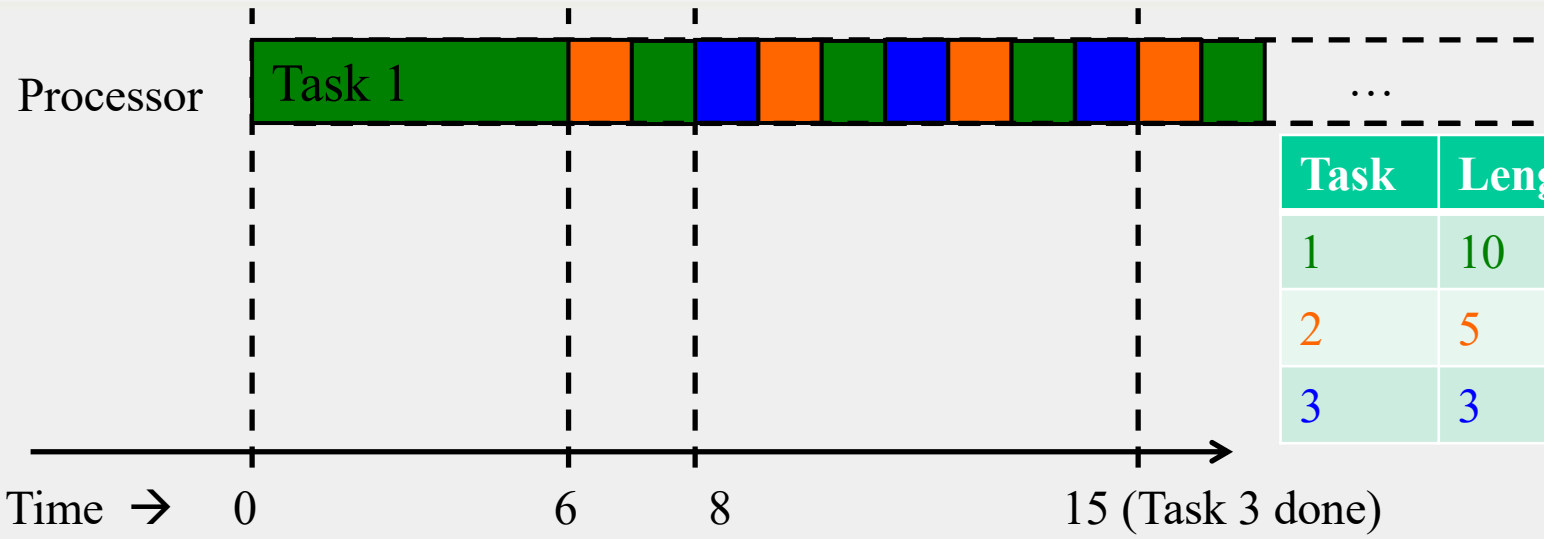
  (versus 14.33 for FIFO/FCFS)



- In general, STF is a special case of *priority scheduling*
  - Instead of using time as priority, scheduler could use user-provided priority

Round-Robin Scheduling

What if you set the time slice to be too low?



Task	Length	Arrival
1	10	0
2	5	6
3	3	8

- *Use a quantum (say 1 time unit) to run portion of task at queue head*
- *Pre-empts processes by saving their state, and resuming later*
- *After pre-empting, add to end of queue*

Round-Robin vs. STF/FIFO

- Round-Robin preferable for
  - Interactive applications
  - User needs quick responses from system
- FIFO/STF preferable for Batch applications
  - User submits jobs, goes away, comes back to get result

Summary

- Single processor scheduling algorithms
  - FIFO/FCFS
  - Shortest task first (optimal!)
  - Priority
  - Round-robin
  - Many other scheduling algorithms out there!
- Next: what about cloud scheduling?

Hadoop Scheduling

Hadoop Scheduling

(Hadoop 2.0)

- A Hadoop job consists of Map tasks and Reduce tasks
- Only one job in entire cluster => it occupies cluster
- Multiple customers with multiple jobs
  - Users/jobs = "tenants"
  - Multi-tenant system
- => Need a way to schedule all these jobs (and their constituent tasks)
- => Need to be *fair* across the different tenants
- Hadoop YARN has two popular schedulers
  - *Hadoop Capacity Scheduler*
  - *Hadoop Fair Scheduler*

Hadoop Capacity Scheduler

- Contains multiple queues
- Each queue contains multiple jobs
- Each queue guaranteed some portion of the cluster capacity

  E.g.,
  - Queue 1 is given 80% of cluster
  - Queue 2 is given 20% of cluster
  - Higher-priority jobs go to Queue 1
- For jobs within same queue, FIFO typically used
  - Job can occupy the entire share of cluster
  - If not enough tasks, next job can be started to occupy the rest
- Administrators can configure queues

Source: http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

Elasticity in HCS



Advantage of not pre-empting tasks?

- Administrators can configure each queue with limits
  - Soft limit: how much % of cluster is the queue guaranteed to occupy
  - (Optional) Hard limit: max % of cluster given to the queue
- Elasticity
  - A queue can consume (borrow) the unused cluster resources of another queue
  - But those resources are returned if new jobs arrive to that other queue
- How do resources get returned?
  - By default, pre-emption not allowed: wait until tasks complete
  - Can enable pre-emption: pick queue, then job, and then tasks of that job
    - Typically, LIFO. Why?

Other HCS Features

- Queues can be hierarchical
  - May contain child sub-queues, which may contain child sub-queues, and so on
  - Child sub-queues can share resources equally
- Scheduling can take memory requirements into account (memory specified by user)

Hadoop Fair Scheduler

- Goal: all jobs get equal share of resources
- When only one job present, occupies entire cluster
- As other jobs arrive, each job given equal % of cluster
  - E.g., Each job might be given equal number of cluster-wide YARN containers
  - Each container == 1 task of job

Source: http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html

Hadoop Fair Scheduler (2)

- Divides cluster into pools
  - Typically, one pool per user
- Resources divided equally among pools
  - Gives each user fair share of cluster
- Within each pool, can use either
  - Fair share scheduling, or
  - FIFO/FCFS
  - (Configurable)

Pre-emption in HFS



Why is okay to re-execute tasks?

- Some pools may have *minimum shares*
  - Minimum % of cluster that pool is guaranteed
- When minimum share not met in a pool, for a while
  - Take resources away from other pools
  - By pre-empting jobs in those other pools
  - By *killing* the currently-running tasks of those jobs
    - Tasks can be re-started later
    - Ok since tasks are idempotent!
  - To kill, scheduler picks most-recently-started tasks
    - Minimizes wasted work

Other HFS Features

- Can also set limits on
  - Number of concurrent jobs per user
  - Number of concurrent jobs per pool
  - Number of concurrent tasks per pool
- Prevents cluster from being hogged by one user/job

Estimating Task Lengths

- HCS/HFS use FIFO
  - May not be optimal (as we know!)
  - Why not use shortest-task-first instead? It's optimal (as we know!)
- Challenge: Hard to know expected running time of task (before it's completed)
- Solution: Estimate length of task
- Some approaches
  - Within a job: Calculate running time of task as proportional to size of its input
  - Across tasks: Calculate running time of task in a given job as average of other tasks in that given job (weighted by input size)
- Lots of recent research results in this area!

Summary

- Hadoop Scheduling in YARN
  - Hadoop Capacity Scheduler
  - Hadoop Fair Scheduler

- Thus far we've considered only one type of resource
  - Either processor, or memory
  - How about multi-resource requirements?
  - Next!

Dominant-Resource Fair Scheduling

Challenge

- What about scheduling VMs in a cloud (cluster)?
- Difference from prior scheduling mechanisms
  - Jobs/tasks don't have lengths, but resource specifications
- Jobs may have multi-resource requirements
  - Job 1's tasks: 2 CPUs, 8 GB
  - Job 2's tasks: 6 CPUs, 2 GB
- How do you schedule these jobs in a "fair" manner?
  - What does fairness even mean?
  - That is, how many tasks of each job do you allow the system to run concurrently?

Dominant Resource Fairness (DRF)

- Proposed by researchers from U. California Berkeley
- Proposes notion of fairness across jobs with multi-resource requirements
- DRF is
  - Fair for multi-tenant systems
  - Strategy-proof: tenant can't benefit by lying
  - Envy-free: tenant can't envy another tenant's allocations

Where is DRF Useful?

- DRF is
  - Usable in scheduling VMs in a cluster
  - Available as an option for scheduling in Hadoop YARN
- DRF used in Mesos, an OS intended for cloud environments
- DRF-like strategies also used by cloud computing vendors inside their managed customer-facing container and big data platforms

How DRF Works

- Our example
  - Job 1's tasks: 2 CPUs, 8 GB

    => Job 1's *resource vector* = <2 CPUs, 8 GB>
  - Job 2's tasks: 6 CPUs, 2 GB

    => Job 2's *resource vector* = <6 CPUs, 2 GB>
- Consider a cloud with <18 CPUs, 36 GB RAM>

How DRF Works (2)

- Our example
  - Job 1's tasks: 2 CPUs, 8 GB
    => Job 1's *resource vector* = <2 CPUs, 8 GB>
  - Job 2's tasks: 6 CPUs, 2 GB
    => Job 2's *resource vector* = <6 CPUs, 2 GB>
- Consider a cloud with <18 CPUs, 36 GB RAM>
- Each Job 1's task consumes % of total CPUs = 2/18 = **1/9**
- Each Job 1's task consumes % of total RAM = 8/36 = **2/9**
- 1/9 < 2/9
  - => *Job 1's dominant resource is RAM*, i.e., Job 1 is more memory-intensive than it is CPU-intensive

How DRF Works (3)



What is Job2's dominant resource if the cloud had 120 CPUs instead?

- Our example
  - Job 1's tasks: 2 CPUs, 8 GB
    => Job 1's *resource vector* = <2 CPUs, 8 GB>
  - Job 2's tasks: 6 CPUs, 2 GB
    => Job 2's *resource vector* = <6 CPUs, 2 GB>
- Consider a cloud with <18 CPUs, 36 GB RAM>
- Each Job 2's task consumes % of total CPUs = 6/18 = 6/18
- Each Job 2's task consumes % of total RAM = 2/36 = 1/18
- 6/18 > 1/18
  - => *Job 2's dominant resource is CPU*, i.e., Job 2 is more CPU-intensive than it is memory-intensive

DRF Fairness

- DRF equalizes the share of each job's dominant resource across the cluster
  - Job 1's % of RAM = Job 2's % of CPU
- Can be written as linear equations, and solved

DRF Solution, For our Example



Will resources always be fully utilized?

- DRF Ensures
  - Job 1's % of RAM = Job 2's % of CPU
- Solution for our example:
  - Job 1 gets 3 tasks each with <2 CPUs, 8 GB>
  - Job 2 gets 2 tasks each with <6 CPUs, 2 GB>
  - Job 1's % of RAM

= Number of tasks * RAM per task / Total cluster RAM

= 3*8/36 = 2/3

  - Job 2's % of CPU

= Number of tasks * CPU per task / Total cluster CPUs

= 2*6/18 = 2/3

Other DRF Details

- DRF generalizes to multiple jobs
- DRF also generalizes to more than 2 resource types
  - CPU, RAM, Network, Disk, etc.
- DRF ensures that each job gets a fair share of that type of resource which the job desires the most
  - Hence fairness

Internal Cluster Scheduling at Cloud Vendors

- Cloud vendors use proprietary cluster-management and orchestration infrastructure
  - Google has published about theirs: Borg (now called Omega)
  - AWS, Azure, etc. have their equivalents
- Some combination of per-tenant quote/allocations with priority-based or multi-Q scheduling
  - Paired with some version of Round-robin or FIFO within jobs of the same priority
  - "Hierarchy" of schedulers: global, regional, per "cell", etc.
  - Typically, a leader per "cell" coordinates with per-machine daemon
  - Daemon may have ways to give hints the OS scheduler on each machine

Summary: Scheduling

- Scheduling very important problem in cloud computing
  - Limited resources, lots of jobs requiring access to these resources
- Single-processor scheduling
  - FIFO/FCFS, STF, Priority, Round-Robin
- Hadoop scheduling
  - Capacity scheduler, Fair scheduler
- Dominant-Resources Fairness