

CS 425 / ECE 428
Distributed Systems
Fall 2016

Indranil Gupta (Indy)

Nov 10, 2016

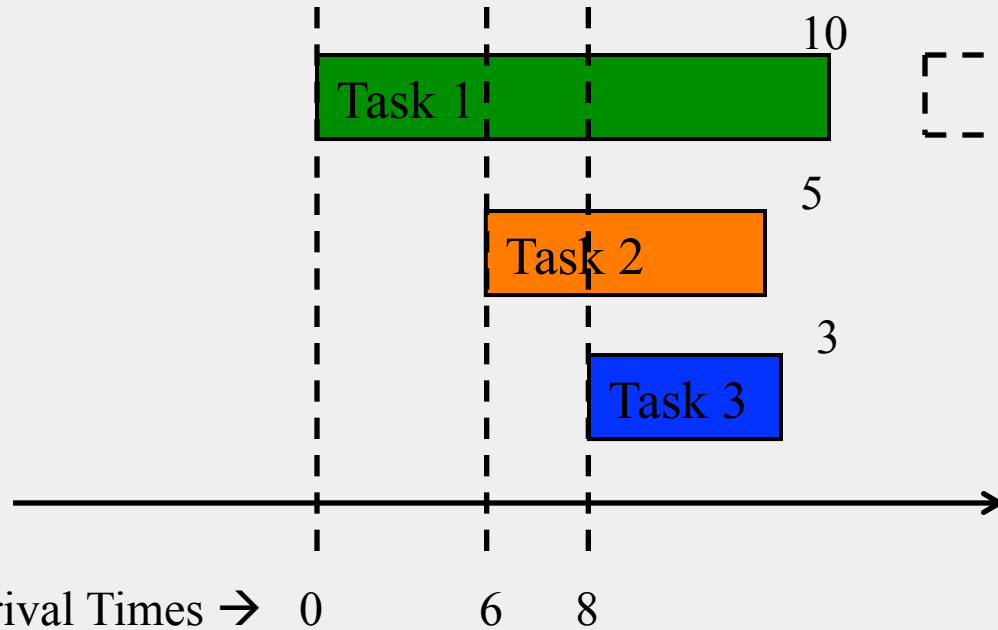
Lecture 24: Scheduling

All slides © IG

WHY SCHEDULING?

- Multiple “tasks” to schedule
 - The processes on a single-core OS
 - The tasks of a Hadoop job
 - The tasks of multiple Hadoop jobs
- Limited resources that these tasks require
 - Processor(s)
 - Memory
 - (Less contentious) disk, network
- Scheduling goals
 1. Good throughput or response time for tasks (or jobs)
 2. High utilization of resources

SINGLE PROCESSOR SCHEDULING

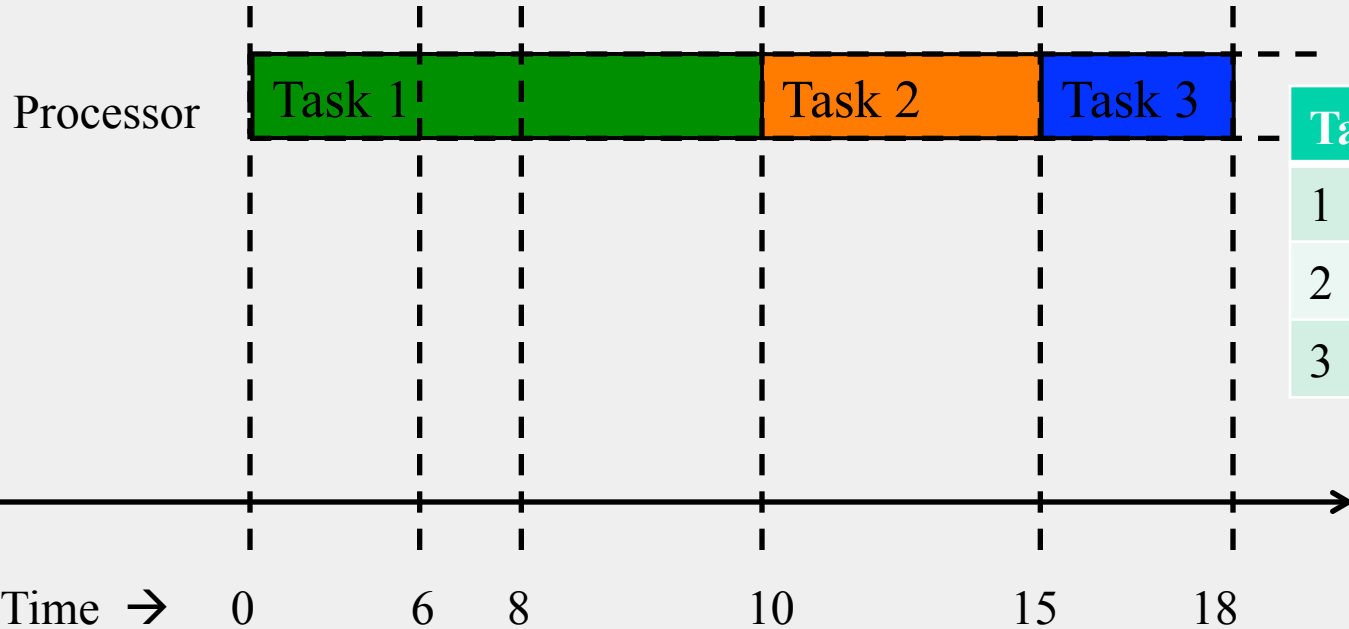


Which tasks run when?

Processor

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

FIFO SCHEDULING (FIRST-IN FIRST-OUT)/FCFS



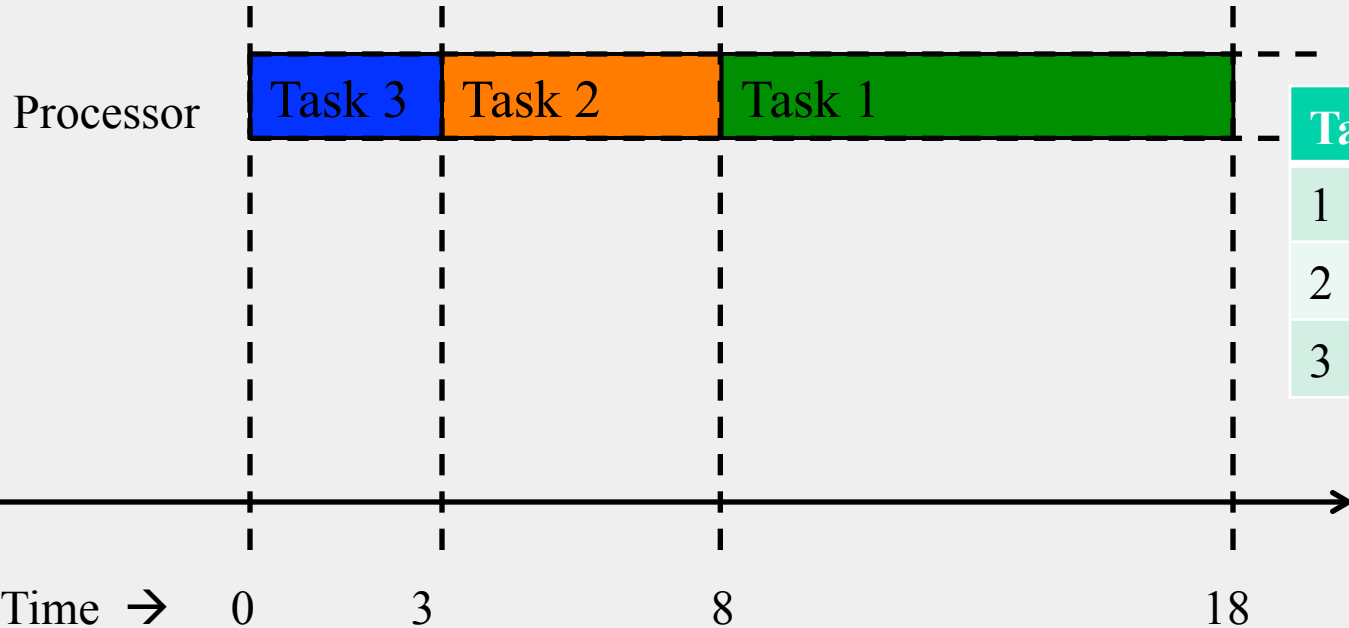
Task	Length	Arrival
1	10	0
2	5	6
3	3	8

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

FIFO/FCFS PERFORMANCE

- Average completion time may be high
- For our example on previous slides,
 - Average completion time of FIFO/FCFS =
(Task 1 + Task 2 + Task 3)/3
= (10+15+18)/3
= 43/3
= 14.33

STF SCHEDULING (SHORTEST TASK FIRST)



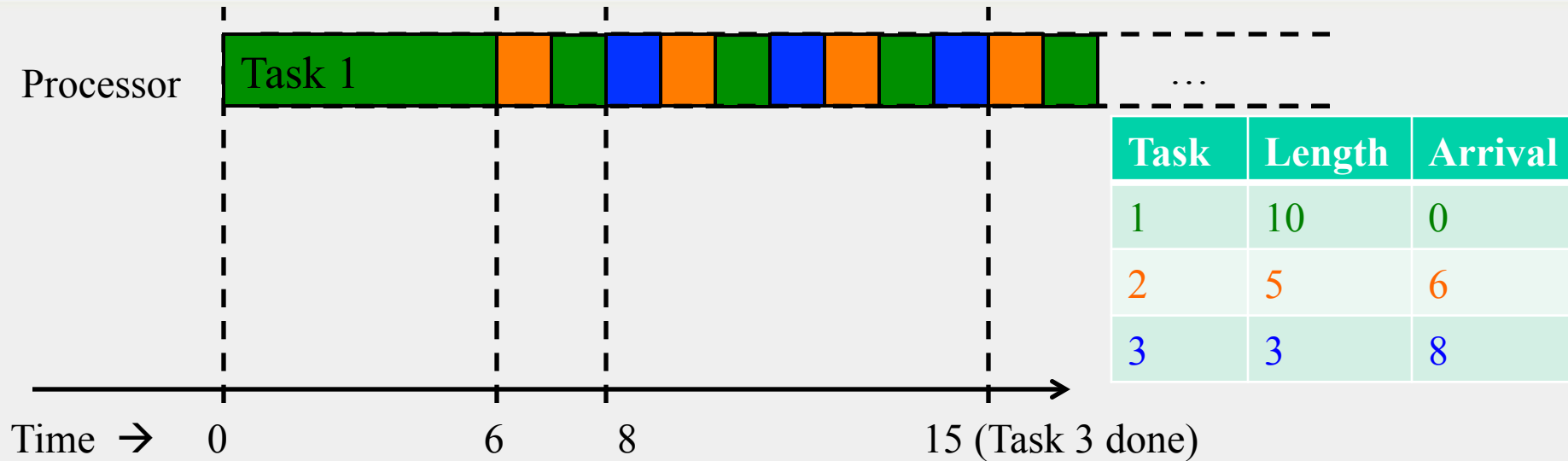
Task	Length	Arrival
1	10	0
2	5	0
3	3	0

- *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



- 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!
- What about cloud scheduling?
 - Next!

Hadoop Scheduling

HADOOP SCHEDULING

- 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
- Administrators can configure queues

ELASTICITY IN HCS

- 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 allowed to occupy more of cluster if resources free
 - But if other queues below their capacity limit, now get full, need to give these other queues resources
- Pre-emption not allowed!
 - Cannot stop a task part-way through
 - When reducing % cluster to a queue, wait until some tasks of that queue have finished

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

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

- 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

- Yet, so far we've talked of only one kind 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)?
- 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?
- That is, how many tasks of each job do you allow the system to run concurrently?
- What does fairness even mean?

DOMINANT RESOURCE FAIRNESS (DRF)

- Proposed by researchers from U. California Berkeley
- Proposes notion of fairness across jobs with multi-resource requirements
- They showed that 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
 - Usable in scheduling Hadoop in a cluster
- DRF used in Mesos, an OS intended for cloud environments
- DRF-like strategies also used some cloud computing company's distributed OS's

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)

- 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 1 is more CPU-intensive than it is memory-intensive

DRF FAIRNESS

- For a given job, the % of its dominant resource type that it gets cluster-wide, is the same for all jobs
 - Job 1's % of RAM = Job 2's % of CPU
- Can be written as linear equations, and solved

DRF SOLUTION, FOR OUR EXAMPLE

- 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

SUMMARY: SCHEDULING

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

ANNOUNCEMENTS