Cluster Scheduling

COS 418: *Distributed Systems*
Lecture 23

Michael Freedman

[Heavily based on content from Ion Stoica]

Key aspects of cloud computing

1. Illusion of infinite computing resources available on demand, eliminating need for up-front provisioning
2. The elimination of an up-front commitment
3. The ability to pay for use of computing resources on a short-term basis

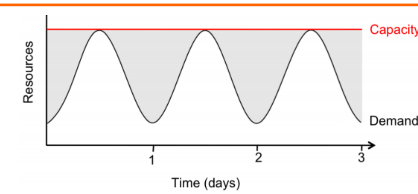
From "Above the Clouds: A Berkeley View of Cloud Computing"

Two main sources of resource demand

- "Services"
  - External demand, scale supply to match demand
- "Data analysis"
  - Tradeoff scale & completion time
    - E.g., use 1 server for 10 hours vs. 10 servers for 1 hour
  - Source of demand elasticity!

Type of contract	Price (m4.xlarge)
Spot - 1 hr duration	$0.139 / hour
Spot– 6 hr duration	$0.176 / hour
On-demand	$0.215 / hour

Towards fuller utilization



- Source of variable demand?
  - Search, social networks, e-commerce, usage have diurnal patterns
  - Apocryphal story: AWS exists because Amazon needed to provision for holiday shopping season, wanted to monetize spare capacity
- But…if provision for peak, what around remaining time?
  - Fill-in with non-time-sensitive usage, e.g., various data crunching
  - E.g., Netflix using AWS at night for video transcoding

Today's lecture

- Metrics / goals for scheduling resources
- System architecture for big-data scheduling

Scheduling: An old problem

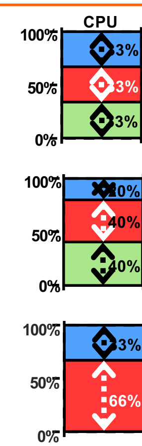
- **CPU allocation**
  – Multiple processors want to execute, OS selects one to run for some amount of time
- **Bandwidth allocation**
  – Packets from multiple incoming queue want to be transmitted out some link, switch chooses one

What do we want from a scheduler?

- **Isolation**
  – Have some sort of guarantee that misbehaved processes cannot affect me "too much"
- **Efficient resource usage**
  – Resource is not idle while there is process whose demand is not fully satisfied
  – "*Work conservation*" -- not achieved by hard allocations
- **Flexibility**
  – Can express some sort of priorities, e.g., strict or time based

Single Resource: Fair Sharing

- *n* users want to share a resource (e.g. CPU)
  – Solution: give each *1/n* of the shared resource
- Generalized by ***max-min fairness***
  – Handles if a user wants less than its fair share
  – E.g. user 1 wants no more than 20%
- Generalized by ***weighted max-min fairness***
  – Give weights to users according to importance
  – User 1 gets weight 1, user 2 weight 2



Max-Min Fairness is Powerful

- **Weighted Fair Sharing / Proportional Shares**
  - User u1 gets weight 2, u2 weight 1
- **Priorities:** Give u1 weight 1000, u2 weight 1
- **Reservations**
  - Ensure u1 gets 10%: Give u1 weight 10, sum weights ≤ 100
- **Deadline-based scheduling**
  - Given a job's demand and deadline, compute user's reservation / weight
- **Isolation:** Users cannot affect others beyond their share

9

Max-min Fairness via Fair Queuing

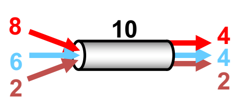
- Fair queuing explained in a fluid flow system: reduces to bit-by-bit round robin among flows
  - Each flow receives $min(r_i, f)$, where
    - r_i – flow arrival rate
    - f – link fair rate (see next slide)
- Weighted Fair Queuing (WFQ)
  - Associate a weight with each flow

10

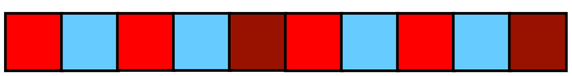
Fair Rate Computation

- If link congested, compute f such that

$$\sum_i \min(r_i, f) = C$$



f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

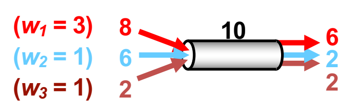


11

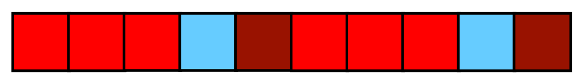
Fair Rate Computation

- Associate a weight w_i with each flow i
- If link congested, compute f such that

$$\sum_i \min(r_i, f \times w_i) = C$$



f = 2:
min(8, 2*3) = 6
min(6, 2*1) = 2
min(2, 2*1) = 2



12

Theoretical Properties of Max-Min Fairness

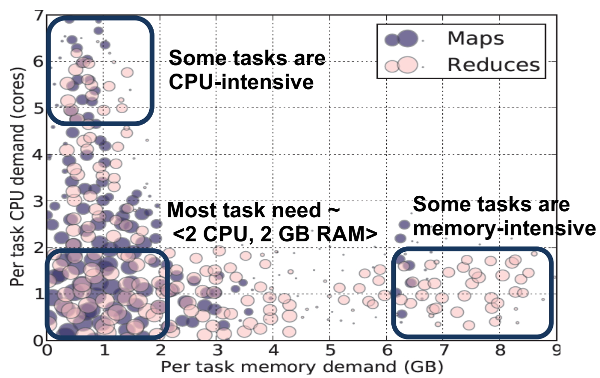
- **Share guarantee**
  - Each user gets at least 1/n of the resource
  - But will get less if her demand is less
- **Strategy-proof**
  - Users are not better off by asking for more than they need
  - Users have no reason to lie

Why is Max-Min Fairness Not Enough?

- Job scheduling is not only about a ***single*** resource
  - Tasks consume CPU, memory, network and disk I/O


- What are task demands today?

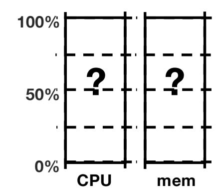
Heterogeneous Resource Demands



2000-node Hadoop Cluster at Facebook (Oct 2010)

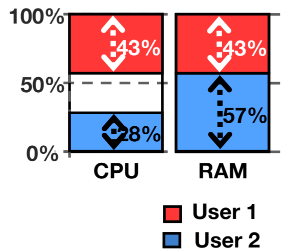
How to allocate?

- 2 resources: CPUs & memory
- User 1 wants **<1 CPU, 4 GB>** per task
- User 2 wants **<3 CPU, 1 GB>** per task
- ***What's a fair allocation?***



A Natural Policy

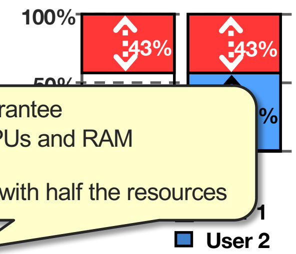
- ***Asset Fairness:*** Equalize each user's ***sum of resource shares***
- Cluster with 28 CPUs, 56 GB RAM
  - U_1 needs <1 CPU, 2 GB RAM> per task, or <3.6% CPUs, 3.6% RAM> per task
  - U_2 needs <1 CPU, 4 GB RAM> per task, or <3.6% CPUs, 7.2% RAM> per task



- Asset fairness yields
  - U_1: 12 tasks: <43% CPUs, 43% RAM> (∑=86%)
  - U_2: 8 tasks: <28% CPUs, 57% RAM> (∑=86%)

Strawman for asset fairness

- Approach: Equalize each user's ***sum of resource shares***
- Cluster with 28 CPUs, 56 GB RAM
  - U_1 needs <1 CPU, 2 GB RAM> per task, or <
  - U_2 n or <



Problem: violates share guarantee
User 1 has < 50% of both CPUs and RAM

Better off in separate cluster with half the resources

- Asset fairness yields
  - U_1: 12 tasks: <43% CPUs, 43% RAM> (∑=86%)
  - U_2: 8 tasks: <28% CPUs, 57% RAM> (∑=86%)

Cheating the Scheduler

- Users willing to *game* the system to get more resources
- Real-life examples
  - A cloud provider had quotas on map and reduce slots
    Some users found out that the map-quota was low.
    Users implemented maps in the reduce slots!
  - A search company provided dedicated machines to users that could ensure certain level of utilization (e.g. 80%).
    Users used busy-loops to inflate utilization.
- How achieve **share guarantee** + **strategy proofness** for sharing?
  - Generalize max-min fairness to multiple resources/

Dominant Resource Fairness (DRF)

- A user's *dominant resource* is resource user has biggest share of
  - Example:

Total resources:	8 CPU	5 GB
User 1's allocation:	2 CPU	1 GB
	25% CPUs	**20% RAM**

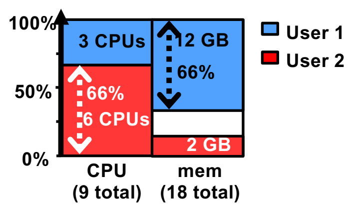
Dominant resource of User 1 is CPU (as 25% > 20%)

- A user's *dominant share:* fraction of dominant resource allocated
  - User 1's dominant share is **25%**

Dominant Resource Fairness: Fair Allocation of Multiple Resource Types
Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica, NSDI'11

Dominant Resource Fairness (DRF)

- Apply max-min fairness to dominant shares
- Equalize the dominant share of the users. Example:
  - Total resources: **<9 CPU, 18 GB>**
  - User 1 demand: **<1 CPU, 4 GB>;** dom res: **mem** (1/9 < 4/18)
  - User 2 demand: **<3 CPU, 1 GB>;** dom res: **CPU** (3/9 > 1/18)



<u>Online DRF Scheduler</u>

Whenever available resources and tasks to run:

Schedule task to user with smallest dominant share

Today's lecture

1. Metrics / goals for scheduling resources
2. System architecture for big-data scheduling

Many Competing Frameworks

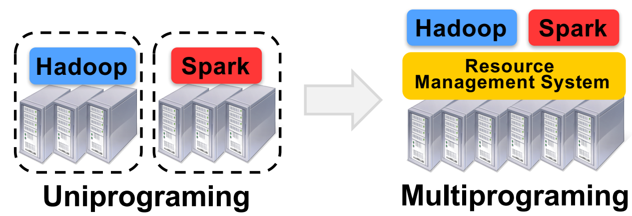
- Many different "Big Data" frameworks
  - Hadoop | Spark
  - Storm | Spark Streaming | Flink
  - GraphLab
  - MPI
- Heterogeneity will rule
  - No single framework optimal for all applications
  - So…each framework runs on dedicated cluster?

One Framework Per Cluster Challenges

- Inefficient resource usage
  - E.g., Hadoop cannot use underutilized resources from Spark
  - Not work conserving
- Hard to share data
  - Copy or access remotely, expensive
- Hard to cooperate
  - E.g., Not easy for Spark to use graphs generated by Hadoop

Common resource sharing layer ?

- Abstracts ("virtualizes") resources to frameworks
- Enable diverse frameworks to share cluster
- Make it easier to develop and deploy new frameworks



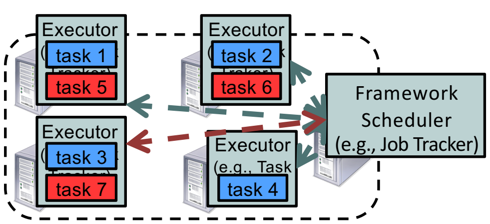
Abstraction hierarchy 101

In a cluster:

… a ***framework*** (e.g., Hadoop, Spark) manages 1+ ***jobs***

… a ***job*** consists of 1+ ***tasks***

… a ***task*** (e.g., map, reduce) involves 1+ processes executing on single machine



Job 1: tasks 1, 2, 3, 4

Job 2: tasks 5, 6, 7

Abstraction hierarchy 101

In a cluster:

… a ***framework*** (e.g., Hadoop, Spark) manages 1+ ***jobs***

… a ***job*** consists of 1+ ***tasks***

… a ***task*** (e.g., map, reduce) involves 1+ processes executing on single machine

- Seek fine-grained resource sharing
  - Tasks typically short: median ~= 10 sec – minutes
  - Better data locality / failure-recovery if tasks fine-grained

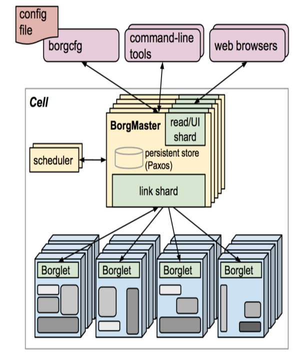
Approach #1: Global scheduler

- Global scheduler takes input, outputs task schedule
  - **Organization policies**
  - **Resource Availability**
  - **Estimates:** Task durations, input sizes, xfer sizes, …
  - **Job requirements:** Latency, throughput, availability…
  - **Job execution plan:** Task DAG, inputs/outups
- Advantages: “Optimal”
- Disadvantages
  - More complex, harder to scale (yet Google: 10,000s servers/scheduler
  - Anticipate future requirements, refactor existing

Google’s Borg

- Centralized Borgmaster + Localized Borglet (manage/monitor tasks)
- Goal: Find machines for a given job

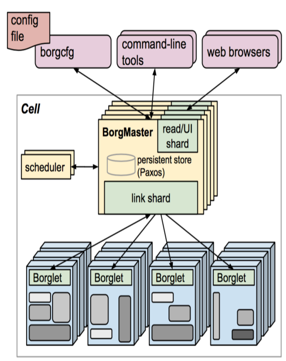
```
job hello = {
  runtime = { cell = “ic” }
  binary = ‘../hello_webserver’
  args = { port = ‘%port%’ }
  requirements = {
    RAM = 100M
    disk = 100M
    CPU = 0.1
  }
  replicas = 10000
}
```



Large-scale cluster management at Google with Borg
A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, EuroSys 15

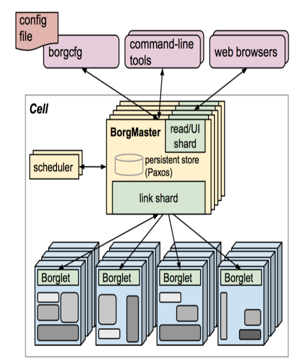
Google’s Borg

- Centralized Borgmaster + Localized Borglet (manage/monitor tasks)
- Goal: Find machines for a given job
- Used across all Google services
  - Services: Gmail, web search, GFS
  - Analytics: MapReduce, streaming
    - Framework controller sends master allocation request to Borg for full job



Google’s Borg

- Centralized Borgmaster + Localized Borglet (manage/monitor tasks)
- Goal: Find machines for a given job
- Allocation
  - Minimize # / priority preempted tasks
  - Pick machines already having copy of the task’s packages
  - Spread over power/failure domains
  - Mix high/low priority tasks



Approach #2: Offers, not schedule

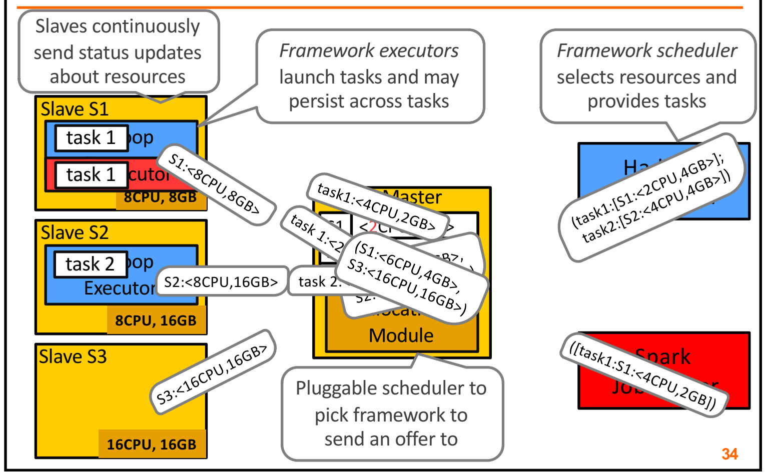
- Unit of allocation: ***resource offer***
  - Vector of available resources on a node
  - E.g., node1: <1CPU, 1GB>, node2: <4CPU, 16GB>

1. Master sends resource offers to frameworks
2. Frameworks:
   - Select which offers to accept
   - Perform task scheduling
   - Unlike global scheduler, requires another level of support

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center
Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica, NSDI'11

33

Mesos Architecture: Example



34

Why does it Work?

- A framework can wait for offer that matches its constraints or preferences, reject otherwise
- Example: Hadoop's job input is *blue* file

35

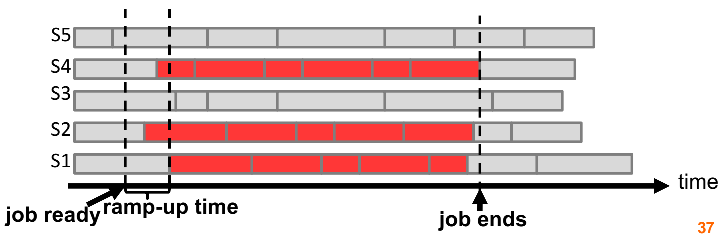
Two Key Questions

- How long does a framework need to wait?
  - Depends on distribution of task duration
  - "Pickiness" of framework given hard/soft constraints
- How allocate resources of different types?
  - Use DRF!

36

Ramp-Up Time

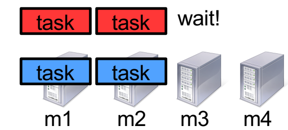
- **Ramp-Up Time**: time job waits to get its target allocation
- Example:
  - Job's target allocation, k = 3
  - Number of nodes job can pick from, n = 5



Improving Ramp-Up Time?

- **Preemption:** preempt tasks
- **Migration:** move tasks around to increase choice:

Job 1 constraint set = {m1, m2, m3, m4}
Job 2 constraint set = {m1, m2}



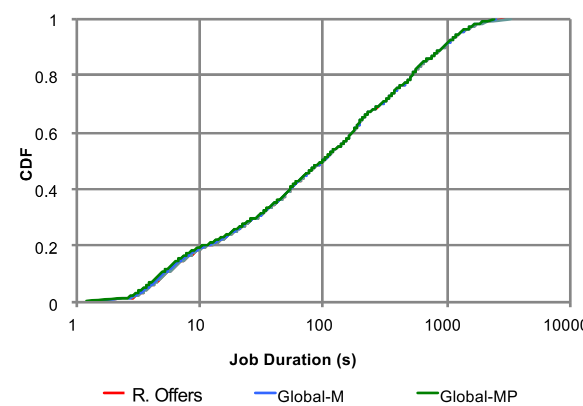
- Existing frameworks implement
  - **No** migration: expensive to migrate short tasks
  - Preemption with task killing (e.g., Dryad's Quincy): expensive to checkpoint data-intensive tasks

Macro-benchmark

- Simulate an 1000-node cluster
  - Job and task durations: Facebook traces (Oct 2010)
  - Constraints: modeled after Google*
- Allocation policy: fair sharing
- Scheduler comparison
  - **Resource Offers**: no preemption, and no migration (e.g., Hadoop's Fair Scheduler + constraints)
  - **Global-M**: global scheduler with migration
  - **Global-MP**: global scheduler with migration and preemption

* Sharma et al., "Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters", ACM SoCC, 2011.

Facebook: Job Completion Times



How to allocate resources? DRF!

	CPU	Memory
Cluster Supply	10	20
A's Demand	4 (**40%**)	2 (10%)
B's Demand	1 (10%)	5 (**25%**)

Cluster: Remaining	Cluster: Offer	A's Allocation	B's Allocation
(10cpu, 20gb)	(2cpu, 2gb) to A	✔ **(2cpu, 2gb, 20%)**	(0cpu, 0gb, 0%)
(8cpu, 18gb)	(1cpu, 2gb) to B	(2cpu, 2gb, 20%)	✔ **(1cpu, 2gb, 10%)**
(7cpu, 16gb)	(1cpu, 3gb) to B	(2cpu, 2gb, 20%)	✔ **(2cpu, 5gb, 25%)**
(6cpu, 13gb)	(1cpu, 6gb) to A	✘ (2cpu, 2gb, 20%)	(2cpu, 5gb, 25%)
(6cpu, 13gb)	(1cpu, 6gb) to B	(2cpu, 2gb, 20%)	✔ **(3cpu, 11gb, 55%)**
(5cpu, 7gb)	(3cpu, 2gb) to B	✔ **(5cpu, 4gb, 50%)**	(3cpu, 11gb, 55%)

Today's lecture

- Metrics / goals for scheduling resources
  - Max-min fairness, weighted-fair queuing, DRF
- System architecture for big-data scheduling
  - Central allocator (Borg), two-level resource offers (Mesos)