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!

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

<table>
<thead>
<tr>
<th>Type of contract</th>
<th>2016 Price (m4.xlarge) $/hour</th>
<th>2017 Price (m4.xlarge) $/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spot - 1 hr duration</td>
<td>$0.139</td>
<td>$0.10</td>
</tr>
<tr>
<td>Spot - 6 hr duration</td>
<td>$0.176</td>
<td>$0.13</td>
</tr>
<tr>
<td>On-demand</td>
<td>$0.215</td>
<td>$0.20</td>
</tr>
</tbody>
</table>
Today’s lecture

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

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

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

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

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

Fair Rate Computation

- If link congested, compute \( f \) such that
  \[
  \sum_i \min(r_i, f) = C
  \]
  \[
  f = 4: \\
  \min(8, 4) = 4 \\
  \min(6, 4) = 4 \\
  \min(2, 4) = 2
  \]

Fair Rate Computation

- Associate a weight \( w_i \) with each flow \( i \)
- If link congested, compute \( f \) such that
  \[
  \sum_i \min(r_i, f \times w_i) = C
  \]
  \[
  f = 2: \\
  \min(8, 2 \times 3) = 6 \\
  \min(6, 2 \times 1) = 2 \\
  \min(2, 2 \times 1) = 2
  \]
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
  - U₁ needs <1 CPU, 2 GB RAM> per task, or <3.6% CPUs, 3.6% RAM> per task
  - U₂ needs <1 CPU, 4 GB RAM> per task, or <3.6% CPUs, 7.2% RAM> per task

  • Asset fairness yields
    - U₁: 12 tasks: <43% CPUs, 43% RAM> (∑=86%)
    - U₂: 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/

Strawman for asset fairness

• Approach: Equalize each user’s sum of resource shares

  Cluster with 28 CPUs, 56 GB RAM
  - U₁ needs <1 CPU, 2 GB RAM> per task, or <3.6% CPUs, 3.6% RAM> per task
  - U₂ needs <1 CPU, 4 GB RAM> per task, or <3.6% CPUs, 7.2% RAM> per task

  • 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
    - U₁: 12 tasks: <43% CPUs, 43% RAM> (∑=86%)
    - U₂: 8 tasks: <28% CPUs, 57% RAM> (∑=86%)

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)

Online DRF Scheduler
Whenever available resources and tasks to run:
Schedule task to user with smallest dominant share

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?

Today’s lecture
1. Metrics / goals for scheduling resources
2. System architecture for big-data scheduling
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

- Seek fine-grained resource sharing
  - Tasks typically short: median \( \approx 10 \text{ sec} – \text{minutes} \)
  - 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/outputs

- 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
  }
  ```

- Used across all Google services
  - Services: Gmail, web search, GFS
  - Analytics: MapReduce, streaming
  - Framework controller sends master allocation request to Borg for full 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

---

Large-scale cluster management at Google with Borg
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

How to allocate resources? DRF!

<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>A’s Demand</td>
<td>4 (40%)</td>
</tr>
<tr>
<td>B’s Demand</td>
<td>1 (10%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cluster: Remaining</th>
<th>Cluster: Offer</th>
<th>A’s Allocation</th>
<th>B’s Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10cpu, 20gb)</td>
<td>(2cpu, 2gb) to A</td>
<td>(0cpu, 0gb, 0%)</td>
<td>(0cpu, 0gb, 0%)</td>
</tr>
<tr>
<td>(10cpu, 20gb)</td>
<td>(4cpu, 3gb) to A</td>
<td>(4cpu, 3gb, 40%)</td>
<td>(0cpu, 0gb, 0%)</td>
</tr>
<tr>
<td>(6cpu, 17gb)</td>
<td>(1cpu, 3gb) to B</td>
<td>(4cpu, 3gb, 40%)</td>
<td>(0cpu, 0gb, 0%)</td>
</tr>
<tr>
<td>(5cpu, 12gb)</td>
<td>(1cpu, 5gb) to B</td>
<td>(4cpu, 3gb, 40%)</td>
<td>(1cpu, 5gb, 25%)</td>
</tr>
<tr>
<td>(1cpu, 10gb)</td>
<td>(4cpu, 2gb) to A</td>
<td>(8cpu, 5gb, 80%)</td>
<td>(1cpu, 5gb, 25%)</td>
</tr>
<tr>
<td>(0cpu, 4gb)</td>
<td>(1cpu, 6gb) to B</td>
<td>(8cpu, 5gb, 80%)</td>
<td>(2cpu, 11gb, 55%)</td>
</tr>
</tbody>
</table>

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)