Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Acknowledgements for the materials: Indy Gupta
## Grade distribution

<table>
<thead>
<tr>
<th></th>
<th>3-credit</th>
<th>4-credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>33%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(drop 2 worst HWs)</td>
</tr>
<tr>
<td>Midterms</td>
<td>33%</td>
<td>25%</td>
</tr>
<tr>
<td>Final</td>
<td>33%</td>
<td>25%</td>
</tr>
<tr>
<td>MPs</td>
<td>N/A</td>
<td>33%</td>
</tr>
<tr>
<td>Participation</td>
<td>1%</td>
<td>1%</td>
</tr>
</tbody>
</table>
Grading

• Midterms curving formula (tentative)
  • relative: 80 + 10*(your score – avg_UG_score) / standard_dev
  • We will use max(absolute, relative) to get final score out of 100.
• Relevant Stats:
  • Midterm1: avg_UG_score = 76.119, std_dev = 14.96
  • Midterm2: avg_UG_score = 75.94, std_dev = 13.28
• Multiply the final score (out of 100) for each midterm by:
  • 0.165 for 3-credit students
  • 0.125 for 4-credit students

• Finals will be similarly curved, but has higher weightage.
Grading

• Homeworks will not be curved.
  • For 3-credit students:
    • (sum of all 5 homework scores) * 100 * 0.33 / 200
  • For 4-credit students:
    • (sum of best 3 homework scores) * 100 * 0.16 / 120

• MPs will not be curved.
  • (sum of all four MP scores) * 100 * 0.33 / 330

• Participation score: directly taken from Campuswire
  • if reported score > 100, you get full 1%
  • Else you get (reported score /100)%
    • Bonus for active participation in class.
Grading

• Grading for active in-class participation
  • Will match faces with roster photograph.
  • If you have actively participated in class, and think you look very different from your roster photograph, please email me a more representative photo.
Tentative Grades Cutoff

- Tentative mapping from score to grade (*rough estimate*):
  - Cutoff for B: 80%
  - Bump up a grade for each 4% leap above 80%.
    - B+ 84%, A- 88%, A 92%, A+ 96%
  - Bump down a grade for each 4% leap below 80%
    - B- 76%, C+ 72%, .....

- This is subject to change!
Cloud Scheduling

Single processor scheduling: when should a task start? (order of task)

Cloud/cluster scheduling: additional dimensions

- how many tasks of each job can we run together across the cluster?
  - minimize average job completion time
  - high cluster resource utilization
  - ensure fairness (e.g. across jobs from different users or tenants)
- on which node should we place a given task?
  - data locality, tasks dependencies, minimize inter-task communication latency, etc.
Case-study 1: 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 resources
  • Queue 2 is given 20% of cluster resources
  • (can specify different percentages for different resource types: memory, compute, etc)
  • Percentages based on business agreements with tenants.

• For jobs within same queue, FIFO typically used

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

- Some (higher priority / production) 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 other cluster resources (disk / external services) from being hogged by one user/job
Estimating Task Lengths

- HCS/HFS use FIFO
  - May not be optimal (as we know!)
  - Why not use SRPT or 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

- 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 jobs 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>

• Naïve fairness: each job gets 9 CPUs and 18 GB RAM.
  • How many tasks for each job?
  • Not Pareto-efficient!
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 2 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
  • Assume J1 has x tasks and J2 has y tasks
  • \( \frac{2x}{9} = \frac{6y}{18} \)
  • \( 2x + 6y \leq 18 \)
  • \( 8x + 2y \leq 36 \)
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
Other DRF Details

• DRF may not always **equalize** dominant resource shares.
  • e.g. when a job’s demand is met and does not need more tasks.
  • or if one resource gets exhausted, jobs not using that resource can still be served.
**Algorithm 1** DRF pseudo-code

\[ R = \langle r_1, \cdots, r_m \rangle \quad \triangleright \text{total resource capacities} \]
\[ C = \langle c_1, \cdots, c_m \rangle \quad \triangleright \text{consumed resources, initially 0} \]
\[ s_i \ (i = 1..n) \quad \triangleright \text{user } i\text{'s dominant shares, initially 0} \]
\[ U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ (i = 1..n) \quad \triangleright \text{resources given to user } i, \text{initially 0} \]

**pick** user \( i \) with lowest dominant share \( s_i \)

\( D_i \leftarrow \text{demand of user } i\text{'s next task} \)

**if** \( C + D_i \leq R \) **then**

\( C = C + D_i \quad \triangleright \text{update consumed vector} \)
\[ U_i = U_i + D_i \quad \triangleright \text{update } i\text{'s allocation vector} \]
\[ s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\} \]

**else**

**return**

\( \triangleright \text{the cluster is full} \)

**end if**
DRF Algorithm

Algorithm 1 DRF pseudo-code

\[ R = \langle r_1, \ldots, r_m \rangle \quad \text{\triangleright total resource capacities} \]
\[ C = \langle c_1, \ldots, c_m \rangle \quad \text{\triangleright consumed resources, initially 0} \]
\[ s_i (i = 1..n) \quad \text{\triangleright user i's dominant shares, initially 0} \]
\[ U_i = \langle u_{i,1}, \ldots, u_{i,m} \rangle (i = 1..n) \quad \text{\triangleright resources given to user i, initially 0} \]

pick user \( i \) with lowest dominant share \( s_i \)
\[ D_i \leftarrow \text{demand of user i's next task} \]

if \( C + D_i \leq R \) then
    \[ C = C + D_i \quad \text{\triangleright update consumed vector} \]
    \[ U_i = U_i + D_i \quad \text{\triangleright update i's allocation vector} \]
    \[ s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\} \]
else
    return \quad \text{\triangleright the cluster is full} 
end if

Our example

Job 1's tasks: 2 CPUs, 8 GB
\[ \Rightarrow \text{Job 1's resource vector} = \langle 2 \text{ CPUs}, 8 \text{ GB} \rangle \]
Job 2's tasks: 6 CPUs, 2 GB
\[ \Rightarrow \text{Job 2's resource vector} = \langle 6 \text{ CPUs}, 2 \text{ GB} \rangle \]

Consider a cloud with \langle 18 \text{ CPUs}, 36 \text{ GB RAM} \rangle
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
- Highly active area of research!!