Spring 2022: BDML HW 1

February 16 (Due Feb 23 at 5pm ET)

Name: NetID:

Please submit your work on Gradescope https://www.gradescope.com/courses/371809 by 5pm on February 23.

1 Checking Allocations

Each of the following points describes how resources are allocated between different jobs. **For each, state whether or not the give allocations is *pareto efficient* and whether or not the allocations are *sharing incentive*. Give reasons for your answer.** In each case assume that the cluster has 10 CPUs and 100 GB of RAM, and we can only allocate whole CPUs to each job.

(i) Job A can use as many CPUs and as much memory as it is given, job B can use up to 2 CPUs and 3 GB of memory, and job C can used 1 CPU and 1 GB of RAM. The scheduler allocates job A 4 CPUs and 10 GB of RAM; job B 2 CPUs and 3 GB of RAM; and job C 1 CPU and 1 GB of RAM.

(ii) Job A can use as many CPUs as provided, but requires 20 GB of RAM per CPU. Job B can use as much RAM as provided, but requires 4 CPUs per GB of RAM. The scheduler allocates 4 CPUs and 80GB of RAM to job A, and 4 CPUs and 1GB of RAM to job B,

(iii) Job A can use as many CPUs as provided, but requires 2GB of RAM per CPU. Job B can use as much RAM as provided but requires 8 CPUs per GB or RAM. The scheduler allocates Job A 2 CPUs and 4GB of RAM; and Job B 8 CPUs and 1GB of RAM.

(iv) Job A requires 4 CPUs and 50GB of RAM; job B requires 4 CPUs and 25 GB of RAM. The scheduler allocates Job A 4 CPUs and 50 GB of RAM; and B 4 CPUs and 25 GB of RAM.

2 Using DRF to Allocate Resources

A cluster has 20 CPUs and 75 GB of memory. Three users A, B and C want to schedule tasks on this cluster. The demand vectors for *each* of their tasks are shown below, and each of them would like to schedule as many tasks as possible:

User	Demand Vector
A	<CPU: 1, Memory: 5GB>
B	<CPU: 2, Memory: 1GB>
C	<CPU: 2, Memory: 3GB>

Use DRF to decide how many tasks from each user are scheduled. Show you working.

3 Completely Fair Scheduler

In the lecture we talked about how Linux's Completely Fair Scheduler cannot ensure that allocations are either Pareto Efficient or Envy Free. Why is this the case?

People have suggested that one approach to fixing this problem is to implement *work stealing*. In the approach the scheduling algorithm at each core (i.e., CFS) is extended so that if core c has no runnable tasks available (i.e., if all tasks assigned to the core are blocked), then the scheduler moves a task assigned to another core (that is not actively running) and executes it on core c. Which of sharing incentive, pareto efficiency, and envy freedom does this approach provide? Provide explanations.