1 Plan for Today:

1.1 What is scheduling and why focus on it?
This is mostly going to be an attempt at convincing you that focusing on this one topic does not (in-practice) limit the breadth of this course, but does provide us with an opportunity to dig deeper into how different approaches apply to a single problem.

1.2 Course Mechanics
This is of course what several of you are likely to have concerns about: how do you get graded, how much work is anticipated, etc.

1.3 Scheduling theory: fairness and other concerns
Some information that might help with next week’s papers.

2 What and why scheduling
This is supposed to be a course on big data and machine learning, and we are largely going to be looking at these topics from the lens of scheduling and resource allocation. Obviously leads to many questions that we will try to address here.

2.1 What do we mean by scheduling?
Really, what we are going to cover is both resource allocation and scheduling.

The problem we care about is thus: we want to run a set of tasks, and have some set of resources, e.g., a set of servers, disks, GPUs, etc. Where should each job run and in what order?
Easy to see (see examples in class) that there are many possible answers to this question, and different answers maximize different utilities. For example, some minimize the total time taken to execute, some minimize average response times, others improve efficiency, etc.

As a result, there are many possible algorithms to address these questions, all of which are correct but apply to different circumstances.

2.2 Why is this a good lens to study big data and machine learning?

Two core reasons:

- First, a lot of what has changed is what resources are used, what jobs are run and what metrics need to be met. End up covering all of those through these lens.
- Second, scheduling lends itself to using all of the algorithmic advancements and tools that people care about.

2.2.1 What has changed

- Larger clusters, different performance characteristics from larger machines.
- New hardware, with different constraints.
- New types of applications with different requirements.

Each of these affects how we schedule and allocate resources as we discussed in class.

2.2.2 Tools people have applied to scheduling

See in class discussion.

3 Course Mechanics

3.1 Course staff and office hours

3.2 How to communicate with us:

Technical questions (about the course, projects, etc): Post on Campuswire. Please try and make your questions public, the answers might help other people. Concerns about grades or administrative questions: E-mail apanda@cs.nyu.edu.
3.3 Grading

<table>
<thead>
<tr>
<th>What</th>
<th>Grade Portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four homeworks</td>
<td>30%</td>
</tr>
<tr>
<td>Midterm</td>
<td>10%</td>
</tr>
<tr>
<td>Final</td>
<td>20%</td>
</tr>
<tr>
<td>Final project</td>
<td>25%</td>
</tr>
<tr>
<td>Class participation</td>
<td>15%</td>
</tr>
</tbody>
</table>

30%: 4 homeworks 30%: 2 exams 25%: Final project 15%: Class participation

3.4 Class participation

5%: Discussing things in class. Read the papers before class, come with interesting questions and thoughts. 10%: More substantial contributions to class discussions. Examples include

- An analysis of prior work (often mentioned in the paper) and a deeper explanation of how things have changed.
- An analysis of concurrent work, comparing alternate decisions.
- A simulation showing how an algorithm behaves with different workloads
- An analysis of workloads that might be particularly well suited or not to the systems we discuss.
- An analysis of where the method breaks down.

How to submit/participate: Post on Campuswire before 5pm on day of class. Include something about whether you would like to present your finding in class (ideally < 5 minutes per person) or what me to present them. If you have a simulation, put it on a public git repository accessible to everyone, complete with information on how to use.

Will give feedback on each such comment. You can submit as many times as you want, will grade based on 2 best submissions.

3.5 Final project

This makes up 25% of the grade. The expectation is that you will do something that gets you more exposure to the material and is new. Examples include:
• Implementing and evaluating one of the scheduling algorithms we talk about for an existing system (e.g., Kubernetes, the Spark standalone scheduler, TF learning rate scheduler, some kernel, etc.). The evaluation here can be done using common benchmarks (e.g., DeathStar Bench for Kubernetes, TPC-DS or Big Data Benchmark for Spark, etc.)

• Run some processes or realistic jobs, and collect information about their resource requirements and how scheduling choices impact their performance. Show this using simulations or implementation.

• Propose extensions or changes to a scheduler (e.g., incorporate some form of fairness into Shinjuku’s scheme, extend DRF to handle different types of jobs, etc.), and then evaluate them in simulation or implementation. In this case you should both justify why the workload you use for evaluation is reasonable, and also compare things before and after.

• Implement and reproduce results from a recent scheduling paper (2019 or later) published at NSDI, OSDI, SOSP or SIGCOMM. You should specifically pay attention to any parameters you needed to tune to recreate the results, or any unstated assumptions they make.

You should pick one of these and need to submit a proposal by February 23.

4 Scheduling theory

4.1 Generalized Processor Sharing and Fairness

An idealized model for sharing a single processor or network link. There are several variants of this, including a paper by Demers, Keshav and Shenker from 1989.

Let us first consider the case where we have \( n \) jobs, all of which have infinite demands. Assume we have \( R \) units of resources, a fair split would be allocate each:

\[
\frac{R}{n} \tag{1}
\]

However, in reality jobs might not have infinite demands, and we want to consider a case where each job has demands \( d_0, \ldots, d_n \). In this case a fair allocation is one where:
• No job is given more resources than it requests.

• No other allocation which meets the previous requirement has a larger minimum allocation.

• Adding resources to one allocation must require reducing the allocation of another.

Why is this useful? Provides fairness and work conservation, ensures that everyone gets a reasonable share of resources and no resources are wasted.

One can always wonder how to achieve such an allocation? That is the topic of one of next week’s papers.

4.2 Job Shop Scheduling

GPS and fairness largely look at the question of how to share one resource between multiple jobs. But what about allocating processing resources across multiple machines. This is a problem we need to address when scheduling jobs in clusters and datacenters. This leads to the more general job-shop scheduling problem: how to assign tasks to multiple resources so they meet some desired constraints. We will discuss this later in the class.