DS 100: Principles and Techniques of Data Science **Date: April 26, 2019**

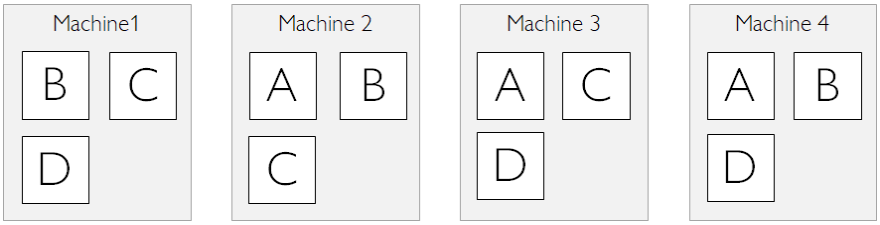
Big Data

Name:

1 Big Data Potpourri [Fall 2018 Final]

1. The figure above from class shows four distinct file blocks labeled A, B, C, and D spread across four machines, where each machine holds exactly 3 blocks.

   (a) For the figure below, at most, how many of our machines can fail without any data loss? 2

   (b) Suppose that instead of 4 machines, we have only 3 machines that can store 3 blocks each. Suppose we want to be able to recover our data even if two machines fail. What is the maximum total number of distinct blocks we can store? 3

   (c) Same as part b, but now suppose we only need to be able to recover our data if one machine fails. What is the maximum total number of distinct blocks we can store? 4

2. For this section, we will be working with the UC Berkeley Undergraduate Career Survey dataset. Each year, the UC Berkeley career center surveys graduating seniors for their plans after graduating. Below is a sample of the full dataset. The full dataset contains many thousands of rows.

   Each record of the `survey` table is an entry corresponding to a student. We have the student's major information (`m_name`), company information (`c_name, c_location`), and the job title (`j_name`).

   Suppose our table has 9,000 rows, with 3,000 unique job names, 1,700 unique company names, 817 unique locations, and 105 unique major names. The table above has many redundancies. Suppose we wanted to instead use the star schema idea from lecture, where we have one fact table and many dimension tables. How many dimension tables would we end up with? How many rows would there be in our fact table? How many columns would there be in our fact table? There may be more than one correct answer.

j_name	c_name	c_location	m_name
Llama Technician	Google	MOUNTAIN VIEW	EECS
Software Engineer	Salesforce	SF	EECS
Open Source Maintainer	Github	SF	Computer Science
Big Data Engineer	Microsoft	REDMOND	Data Science
Data Analyst	Startup	BERKELEY	Data Science
Analyst Intern	Google	SF	Philosophy

Table 1: `survey` Table

i. Number of dimension tables: **4**

ii. Number of rows in fact table: **9000**

iii. Number of columns in fact table: **4**

3. As described in class, the traditional data warehouse is a large tabular database that is periodically updated through the ETL process, which combines data from several smaller data sources into a common tabular format. The alternative is a data lake, where data is stored in its original natural form. Which of the following are good reasons to use a data lake approach?

☐ A. The data is sensitive, e.g. medical data or government secrets.

☐ B. To maximize compatibility with commercial data analysis and visualization tools.

☐ **C. When there is no natural way to store the data in tabular format.**

☐ D. To ensure that the data is clean.

2 Distributed/Parallel Computing

2.1 Primer on ray

2.1.1 Overview

Ray is a distributed execution engine. The same code can be run on a single machine to achieve efficient multiprocessing, and it can be used on a cluster for large computations.

When using Ray, several processes are involved.

1. Multiple **worker** processes execute tasks and store results in object stores. Each worker is a separate process.
2. One **object store** per node stores immutable objects in shared memory and allows workers to efficiently share objects on the same node with minimal copying and deserialization.

3. One **raylet** per node assigns tasks to workers on the same node.
4. A **driver** is the Python process that the user controls. For example, if the user is running a script or using a Python shell, then the driver is the Python process that runs the script or the shell. A driver is similar to a worker in that it can submit tasks to its raylet and get objects from the object store, but it is different in that the raylet will not assign tasks to the driver to be executed.
5. A **Redis server** maintains much of the system's state. For example, it keeps track of which objects live on which machines and of the task specifications (but not data). It can also be queried directly for debugging purposes.

2.1.2 Asynchronous Computation in Ray

Ray enables arbitrary Python functions to be executed asynchronously. This is done by designating a Python function as a **remote function**.

For example, a normal Python function looks like this.

```
def add1(a, b):
    return a + b
```

A remote function looks like this.

```
@ray.remote
def add2(a, b):
    return a + b
```

2.1.3 Remote functions

Whereas calling `add1(1, 2)` returns `3` and causes the Python interpreter to block until the computation has finished, calling `add2.remote(1, 2)` immediately returns an object ID and creates a **task**. The task will be scheduled by the system and executed asynchronously (potentially on a different machine). When the task finishes executing, its return value will be stored in the object store.

```
x_id = add2.remote(1, 2)
ray.get(x_id) # 3
```

The following simple example demonstrates how asynchronous tasks can be used to parallelize computation.

```
import time

def f1():
    time.sleep(1)
```

```
5
6  @ray.remote
7  def f2():
8      time.sleep(1)
9
10 # The following takes ten seconds.
11 [f1() for _ in range(10)]
12
13 # The following takes one second (assuming the system has at least
       ten CPUs).
14 ray.get([f2.remote() for _ in range(10)])
```

2.2 Interview Question: How can we sort large array of numbers in parallel?

Solution: See notebook attached.