Data 101/CS 187: Data Engineering Fall 2026 Discussion 9 – November 13th, 2026

MapReduce, Sampling

Instructor: Aditya Parameswaran *Covers Lecture(s): 19 – 20*

I MapReduce

A MapReduce-style system, such as Apache Hadoop and Apache Spark, requires each stage of computation to be composed of a *map* function and a *reduce* function. The functions have the following type signatures:

$$Map(K, V) \rightarrow \langle K', V' \rangle^*$$
$$Reduce(K', \langle V' \rangle^*) \rightarrow \langle K'', V'' \rangle^*$$

1. *Word count.* Write a MapReduce program to count the appearance of each word in a set of sentences. Specifically, implement the following two functions:
   - The map function should take a sentence as input, split it into words, then output a dictionary of words and their counts in the sentence.
   - The reduce function should take a word, and a list of counts from the map tasks. It should output the total count of this word in all sentences.

```
def map_fn(document: str) -> dict[str, int]:
    word_count = defaultdict(int)
    for word in document.split():
        word_count[word] += 1
    return word_count

def reduce_fn(word: str, counts: list[int]) -> dict[str, int]:
    return {word: sum(counts)}
```

2. Let us reason about how this program will be executed in a distributed MapReduce environment. Suppose the input strings are:
   - hello world
   - hello world mapreduce
   - mapreduce example
   - hello hello hello

   (a) There will be 4 mappers, each processing one sentence. What are their outputs?

   i. {hello: 1, world: 1}
   ii. {hello: 1, world: 1, mapreduce: 1}
   iii. {mapreduce: 1, example: 1}
   iv. {hello: 3}

(b) There will be 4 reducers. The *shuffle* stage, handled by the execution system, will dispatch all the mapper outputs to their corresponding reducers. What will be the inputs to each reducer? For each input value, which mapper will it come from?

i. example: [1] from mapper 3
ii. hello: [1, 1, 3] from mapper 1, mapper 2 and mapper 4 respectively
iii. mapreduce: [1, 1] from mapper 2 and mapper 3 respectively
iv. world: [1, 1] from mapper 1 and mapper 2 respectively

(c) What will be the outputs of the reducers?

- {example: 1}
- {hello: 5}
- {mapreduce: 2}
- {world: 2}

3. **[Challenge]** What if we wanted to compute the relative frequency of each word (its count divided by the total number of words across all sentences)? How might we change our MapReduce program? (Hint: Consider multiple maps/reduces)

In addition to our original MapReduce program that counts the appearance of each word in a set of sentences, if we follow the map/reduce structure, we can have two more stages.

- Stage 2: The map function takes each (word, count) from the previous reduce output, and emits it under a common key "total_words" for aggregation to count the total number of words. The reduce function takes the key "total_words" and outputs the sum of all counts as the total number of words across all documents.
- Stage 3: The map function takes each (word, count) from Stage 1 and the total number of words from Stage 2, divides the word's total count by the total number of words, and outputs the relative frequency of that word. The reduce function takes each (word, relative_frequency) from all map tasks and collects the results into a final dictionary of words and their relative frequencies.

4. Consider the following SQL query to compute the total number of page views of each Wikipedia article. The `daily_page_views` table stores the total views of an article by day, so we want to aggregate them all. How can we implement this in MapReduce style? First write in English what the map and reduce functions should do, then implement these functions.

   ```
   SELECT title, SUM(num_views) FROM daily_page_views GROUP BY title;
   ```

   ```
   def map_fn(title: str, num_views: int) -> dict[str, int]:
       return {title: num_views}

   def reduce_fn(title: str, num_views_list: list[int]) -> dict[str, int]:
       return {title: sum(num_views_list)}
   ```

   The map function takes as input each record (row) of the table, which includes fields such as *title* and *num_views*. For each record, it will output a pair (title, num_views). Then, in the shuffle stage, the system will send the pairs to their corresponding reducers (one reducer for each title).

   In the reduce stage, the reduce function takes as input a *title*, and a list of values (i.e. a list of *num_views* associated with the title). It will sum up the values and output a final *num_views* for each title.

   This seems simple, but the burden is on the execution system, which must execute this at scale. Think 1 million articles, aggregated over 10 years of data. For example, it must figure out:

   - How to run the mappers and reducers over batches of data?
   - How to efficiently manage the *shuffle* operation, which must send the mapper outputs to the correct reducers?
   - How to run this on hundreds of machines, with possible network and software failures?

II Reservoir Sampling

In this section, we will revisit the reservoir sampling algorithm introduced in lecture. Feel free to open the demo to follow along.

5. What is the first step of reservoir sampling assuming the reservoir is of size k?

> The first step is to fill the reservoir with the first k elements from the dataset. This creates an initial sample before the algorithm starts considering whether new incoming elements should replace any of the existing elements in the reservoir.

6. Once the reservoir is filled with the initial k elements, how does the algorithm decide whether to replace an existing element when a new element is encountered?

> For each new element after the initial k, the algorithm generates a random integer j between 0 and the index of the new element. If j is less than k, it replaces the element at index j in the reservoir. This mechanism ensures that every new element has a chance to be included in the reservoir, while maintaining the size of the reservoir at k elements.

7. After processing n elements, what is the probability that each element is in the reservoir?

> The probability is $\frac{k}{n}$. For a proof of this result, see the next question.

8. **[Optional]** Theorem Statement: We are trying to prove that, at any step, each element in the dataset has an equal probability $\frac{k}{n}$ of being in the reservoir, where n is the total number of elements processed so far.

   (a) Base Case: Prove that when $n \leq k$, the probability holds.

> When $n \leq k$, every element can be added directly to the reservoir, since there is space for all of them. Therefore, each element has a probability of $\frac{k}{n} = 1$ of being in the reservoir. This holds because no elements are replaced until the reservoir is full (i.e., when $n = k$).

(b) Induction Hypothesis: Assume that for a reservoir of size k, after n elements have been processed, each element in the dataset has an equal probability of being in the reservoir. What is this probability?

The probability is $\frac{k}{n}$.

(c) Induction Step (considering the $(n+1)$-th element): Show that after processing $n+1$ elements, each element in the dataset still has an equal probability $\frac{k}{n+1}$ of being in the reservoir. Consider an element j that is already in the reservoir after n steps. What are the two cases for element j to remain in the reservoir, and what are their respective probabilities?

There are two cases for element j to remain in the reservoir:

- **Case 1:** The new element $n+1$ is not selected to enter the reservoir. In this case, element j remains in the reservoir by default. The probability of this case is $1 - \frac{k}{n+1}$.
- **Case 2:** The new element $n+1$ is selected to enter the reservoir, but it replaces another element instead of element j. The probability of $n+1$ being selected is $\frac{k}{n+1}$, and if it is selected, the probability that it replaces a different element (not j) is $\frac{k-1}{k}$. The probability of this case is $\frac{k}{n+1} \times \frac{k-1}{k}$

(d) What's the total probability that j remains in the reservoir after processing the $(n+1)$-th element? Hint: combine the probabilities from the previous cases.

Adding the probabilities from Case 1 and Case 2:

$$\left(1 - \frac{k}{n+1}\right) + \frac{k}{n+1} \times \frac{k-1}{k} = \frac{n}{n+1}$$

This confirms the probability that j remains in the reservoir after $n+1$ elements.

(e) Combine the induction step with the induction hypothesis to find the probability that element j is still in the reservoir after $n+1$ steps.

Using the induction hypothesis that each element has a probability $\frac{k}{n}$ of being in the reservoir after n elements, and multiplying by the probability $\frac{n}{n+1}$ that element j remains in the reservoir at step $n+1$:

$$\frac{k}{n} \times \frac{n}{n+1} = \frac{k}{n+1}$$

This completes the induction step, showing that each element in the dataset has an equal probability $\frac{k}{n+1}$ of being in the reservoir after $n+1$ elements.

III Stratified Sampling

9. You are estimating the average commute time for the Bay Area. The population has two groups:

   - Berkeley: 60% of population, mean = 25 mins.
   - SF: 40% of population, mean = 50 mins.

   You collect a sample of $n = 40$ people using the following two approaches:

   - Random Sample: 38 Berkeley, 2 SF
   - Stratified Sampling

   (a) Compute the sample mean for the random sample.

   $\mu_{\text{random}} = (38 \times 25 + 2 \times 50)/40 = 26.25$ mins

   (b) Compute the sample mean for the stratified sample.

   Using a 60:40 split,
   Stratified Sample: 24 Berkeley, 16 SF
   So, $\mu_{\text{stratified}} = 0.6 \times 25 + 0.4 \times 50 = 35$ mins