To understand how the **MapReduce** process works in the context of counting shapes, let's walk through each step with the necessary clarifications, examples, and visualizations. In this assignment, you'll apply the same core principles of **MapReduce** that were explained in the video lecture, but this time with shapes rather than words as the keys.

Step 0: Store the Dataset Across 4 Partitions in HDFS

Before we begin with the MapReduce process itself, the first step involves storing the dataset across 4 partitions in HDFS (Hadoop Distributed File System). In HDFS, data is distributed across multiple nodes to ensure fault tolerance and parallel processing.

- **Partitioning the dataset**:
    - Since there are multiple shapes (squares, stars, circles, hearts, and triangles), the data should be split evenly across the four partitions. The goal is to balance the data such that each partition has an approximately equal distribution of shapes.
    - For example, partitioning might look like this:
        - **Partition 1**: Square, Circle, Triangle, Square
        - **Partition 2**: Star, Heart, Square, Circle
        - **Partition 3**: Heart, Star, Triangle, Star
        - **Partition 4**: Circle, Heart, Square, Triangle

Each partition will hold a subset of the data and will be processed separately. In Hadoop, this partitioning is important for balancing the workload and ensuring efficient processing.

Step 1: Map the Data

The **Map** step involves processing the dataset in parallel and mapping each shape in the dataset to a key-value pair. In this step, we are "mapping" the shapes to a value of **1** because each instance of a shape is counted individually.

Example:

- When the data from **Partition 1** is mapped, we get the following key-value pairs:
    - **Square → 1**
    - **Circle → 1**
    - **Triangle → 1**
    - **Square → 1**

For **Partition 2**, the mapping results in:

- **Star → 1**

- **Heart → 1**
- **Square → 1**
- **Circle → 1**

This is repeated for all four partitions. The key-value pairs are:

- **Key**: Shape (e.g., Square, Star, Circle)
- **Value**: 1 (since each shape occurrence is counted as 1)

Visual representation: Imagine that each partition contains a list of shapes, and these shapes are mapped to key-value pairs like so:

- **Partition 1**: [Square → 1, Circle → 1, Triangle → 1, Square → 1]
- **Partition 2**: [Star → 1, Heart → 1, Square → 1, Circle → 1]
- **Partition 3**: [Heart → 1, Star → 1, Triangle → 1, Star → 1]
- **Partition 4**: [Circle → 1, Heart → 1, Square → 1, Triangle → 1]

Step 2: Sort and Shuffle

The **Sort and Shuffle** step involves sorting and grouping the intermediate key-value pairs produced by the **Map** phase. In this case, we need to group the shapes (keys) together, regardless of the partition they originated from. This ensures that all instances of the same shape are grouped together and can be processed in the next phase.

Example:

- From the four partitions, after shuffling and sorting, the key-value pairs will be grouped by their key (shape):
    - **Square → [1, 1, 1, 1]**
    - **Circle → [1, 1, 1]**
    - **Triangle → [1, 1, 1]**
    - **Star → [1, 1, 1]**
    - **Heart → [1, 1, 1]**

In the **shuffle** step, the system moves the key-value pairs such that all the **Squares** are in one bucket, all the **Circles** in another, and so on. This ensures that for the **Reduce** step, we can calculate the sum of occurrences for each shape.

Visual Representation:

- After sorting and shuffling, the intermediate data might look like this:

- **Group 1: Square → [1, 1, 1, 1]**
- **Group 2: Circle → [1, 1, 1]**
- **Group 3: Triangle → [1, 1, 1]**
- **Group 4: Star → [1, 1, 1]**
- **Group 5: Heart → [1, 1, 1]**

Step 3: Reduce to Calculate the Final Counts

The **Reduce** step involves summarizing the results. In this case, we reduce the key-value pairs by summing the values for each shape. The shape is the key, and the value is the count of that shape.

Example:

- After grouping, the Reduce function will calculate the count of each shape by summing the values:
  - **Square → 4**
  - **Circle → 3**
  - **Triangle → 3**
  - **Star → 3**
  - **Heart → 3**

Each shape has been counted and reduced to its final total.

Final Key-Value Pairs:

- **Square → 4**
- **Circle → 3**
- **Triangle → 3**
- **Star → 3**
- **Heart → 3**

Simplification of the Key-Value Pairs

To simplify the key-value pair representation, instead of writing each key with the value of 1, we will just show the shape. The final output would look like this:

- **Square: 4**
- **Circle: 3**

- **Triangle: 3**
- **Star: 3**
- **Heart: 3**

This simplifies the representation and focuses on the final count.

Conclusion

This exercise demonstrates the core principles of **MapReduce** in action, where data is partitioned, mapped to key-value pairs, sorted and shuffled, and then reduced to final counts. Each shape is treated as a key, and its occurrences are counted through these steps, which is how **MapReduce** can be used to process large datasets efficiently.

By following these steps:

1. **Storing** the dataset across partitions in HDFS,
2. **Mapping** the data into key-value pairs,
3. **Shuffling and Sorting** the keys to group similar shapes together, and
4. **Reducing** the grouped data to calculate the final counts of each shape,

we can perform distributed processing and obtain the desired counts efficiently, even for large datasets. This process highlights the scalability and parallelism that **Hadoop** and **MapReduce** provide for big data analysis.