Department of Computer Engineering
Academic Term : January - May 2024

Class: B.E. (Computer) Semester VIII Division: B

Subject Name: Distributed Systems CSC 801

Name	**Riya Rajesh Patil**
Roll No.	**9282**
Experiment No.	**8**
Experiment Title	**Implement a simplified version of the MapReduce framework for distributed processing**

Aim - Implement a simplified version of the MapReduce framework for distributed processing

```
def map(key, value, map_func):
    """
    Applies the map function to a key-value pair and emits intermediate pairs.
    Args:
    key: The input key.
    value: The input value.
    map_func: The map function to apply.
    Returns:
    A list of intermediate key-value pairs.
    """
    intermediate_pairs = []
    for key_out, value_out in map_func(key, value):
        intermediate_pairs.append((key_out, value_out))
    return intermediate_pairs
```

```
def reduce(key, values, reduce_func):
    """
    Applies the reduce function to a key and its associated values.
    Args:
    key: The intermediate key.
    values: A list of intermediate values for the key.
    reduce_func: The reduce function to apply.
    Returns:
    The final value for the key.
    """
    return reduce_func(key, values)


def mapreduce(data, map_func, reduce_func):
    """
    Executes the MapReduce job on a single machine.
    Args:
    data: An iterable containing key-value pairs.
    map_func: The map function to apply.
    reduce_func: The reduce function to apply.
    Returns:
    A dictionary containing the final results (key-value pairs).
    """
    # Apply map phase
    intermediate_data = {}
    for key, value in data:
        for key_out, value_out in map(key, value, map_func):
            if key_out not in intermediate_data:
                intermediate_data[key_out] = []
```

```
            intermediate_data[key_out].append(value_out)

    # Shuffle & sort (simulated)
    sorted_data = {}
    for key, values in intermediate_data.items():
        sorted_data[key] = sorted(values)

    # Apply reduce phase
    results = {}
    for key, values in sorted_data.items():
        results[key] = reduce(key, values, reduce_func)

    return results

# Example usage

def word_count_map(key, value):
    """
    Map function for word count example.
    """
    for word in value.split():
        yield (word, 1)

def word_count_reduce(key, values):
    """
    Reduce function for word count example.
    """
    return sum(values)
```

data = [("doc1", "my name is riya patil"), ("doc2", "my fathers name is rajesh")]

word_counts = mapreduce(data, word_count_map, word_count_reduce)

print(word_counts)

```
(base) riyapatil@Riyas-Air dc_8 % python map.py
[{'my': 2, 'name': 2, 'is': 2, 'riya': 1, 'patil': 1, 'fathers': 1, 'rajesh': 1}
(base) riyapatil@Riyas-Air dc_8 %
```

Conclusion -

It can be concluded that the MapReduce framework is used for executing parallel data processing tasks on a single machine. It consists of functions for mapping key-value pairs, reducing intermediate values, and orchestrating the entire MapReduce job. The example usage demonstrates a word count application, where the map function tokenizes input text into words and emits key-value pairs, and the reduce function aggregates word counts. For example there were 2 input string given in the program -data = [("doc1", "my name is riya patil"), ("doc2", "my fathers name is rajesh")] since the word count program was written the output is displayed in the form of counted words of both the string together.

Postlab -

1. **Define Distributed System.**
   A distributed system refers to a network of interconnected computers or nodes that work together to achieve a common goal, even though they are physically separate and may be located in different geographical locations. In a distributed system, each node operates independently and communicates with other nodes through messages, enabling them to share resources, coordinate actions, and provide services to users. Distributed systems are designed to handle large-scale computations, data storage, and processing tasks efficiently by distributing the workload across multiple nodes. They offer advantages such as fault tolerance, scalability, and improved performance, but also present challenges related to synchronization, consistency, and network communication. Examples of distributed systems include cloud computing platforms, peer-to-peer networks, and distributed databases, all of which play crucial roles in modern computing environments.

2. **What is transparency?**

Transparency refers to the degree to which the underlying complexities of the system are hidden from users and applications, allowing them to interact with the system as if it were a single, cohesive entity. There are several types of transparency, including access transparency, where users perceive resources as local regardless of their physical location; location transparency, which hides the specific location of resources from users and applications; and failure transparency, where the system handles failures seamlessly without disrupting user interactions. Other types include migration transparency, replication transparency, and concurrency transparency. Transparency enhances the usability and reliability of distributed systems by abstracting away complexity and providing a simplified interface for users and applications to interact with the system.