**PR 1: To understand the overall programming architecture using Map Reduce API in python**

MapReduce is a programming model used for processing large data sets with a distributed algorithm on a cluster. The model is composed of two primary functions: Map and Reduce. Here's an overview of the MapReduce architecture and how it can be implemented in Python:

**1. MapReduce Architecture**

**Map Phase:**

* The Map function processes a key/value pair to generate a set of intermediate key/value pairs.
* The input data is split into smaller chunks, which are processed in parallel by the Map functions.
* The output of the Map function is a set of intermediate key-value pairs.

**Shuffle and Sort Phase:**

* The intermediate data from the Map phase is shuffled and sorted.
* The data is grouped by key so that all values associated with the same key are sent to the same Reducer.

**Reduce Phase:**

* The Reduce function takes the intermediate key-value pairs produced by the Map phase and merges them together to produce a smaller set of values.
* The output of the Reduce function is the final result.

**2. MapReduce in Python**

Python has several libraries and frameworks that can be used to implement the MapReduce model, including mrjob, PySpark, and Hadoop Streaming. Here, I'll provide an example using the mrjob library, which allows you to write and run MapReduce jobs in Python.

**Example using mrjob**

**Installation:** To install mrjob, use pip:

**pip install mrjob**

**MapReduce Job:** Let's consider a simple word count example, where we count the number of occurrences of each word in a set of text files.

**mapreduce\_wordcount.py:**

**from mrjob.job import MRJob**

**from mrjob.step import MRStep**

**class MRWordCount(MRJob):**

**def steps(self):**

**return [**

**MRStep(mapper=self.mapper,**

**reducer=self.reducer)**

**]**

**def mapper(self, \_, line):**

**# Split the line into words**

**for word in line.split():**

**# Yield each word as a key with a count of 1**

**yield (word.lower(), 1)**

**def reducer(self, word, counts):**

**# Sum all the counts for the word**

**yield (word, sum(counts))**

**if \_\_name\_\_ == '\_\_main\_\_':**

**MRWordCount.run()**

**Running the Job:** To run the MapReduce job, use the following command:

**python mapreduce\_wordcount.py input.txt**

Where input.txt is the file containing the text data you want to process.

**Explanation:**

* The MRWordCount class inherits from MRJob, which is the base class for creating MapReduce jobs in mrjob.
* The steps method defines the steps in the MapReduce job. In this case, there's a single step with a mapper and a reducer.
* The mapper method processes each line of input, splits it into words, and yields each word with a count of 1.
* The reducer method takes each word and its associated counts and sums the counts to produce the total count for each word.

**Conclusion**

This is a basic example of using the MapReduce model in Python with the mrjob library. For more complex tasks, you might need to add additional steps or use more advanced features of the library. Other frameworks like PySpark offer more powerful and flexible ways to handle large-scale data processing using the MapReduce model.