

implement simple map reduce program

ChatGPT

ChatGPT

Sure, let's implement a simple Word Count MapReduce program using C++. This program will count the occurrences of each word in a set of documents. This example assumes that the input data is text files with words separated by spaces.

```
`#include <iostream>

#include <fstream>

#include <sstream>

#include <map>

#include <vector>

#include <algorithm>

#include <iterator>

using namespace std;

// Mapper function

void mapFunction(const string& document, map<string, int>& wordCount) {

    istringstream iss(document);

    string word;

    while (iss >> word) {

        // Convert the word to lowercase for case-insensitive counting

        transform(word.begin(), word.end(), word.begin(), ::tolower);

        // Increment the count for the word in the map

        wordCount[word]++;

    }

}

// Reducer function

void reduceFunction(const map<string, int>& wordCount) {

    // Print the word count results
```

```

    for (const auto& entry : wordCount) {

        cout << entry.first << ": " << entry.second << endl;

    }
}

int main() {

    // Read input documents (replace "document1.txt" and "document2.txt" with your file names)

    ifstream file1("document1.txt");

    ifstream file2("document2.txt");

    if (!file1.is_open() || !file2.is_open()) {

        cerr << "Error opening files." << endl;

        return 1;

    }


    // Combine all documents into a single string

    stringstream combinedDocuments;

    combinedDocuments << file1.rdbuf() << " " << file2.rdbuf();

    string allDocuments = combinedDocuments.str();


    // Map phase

    map<string, int> wordCount;

    mapFunction(allDocuments, wordCount);


    // Reduce phase

    reduceFunction(wordCount);

    // Close files

    file1.close();

    file2.close();

```

```
    return 0;  
}
```

This is a simplified version of a MapReduce program. In a real distributed MapReduce system, the map and reduce tasks would be executed across multiple nodes in a cluster. Additionally, error handling and scalability considerations are not addressed in this basic example. This example assumes that the input documents are small enough to fit into memory. In a distributed environment, data would typically be distributed across nodes, and intermediate results would be shuffled and sorted before being passed to the reducer tasks.