Practical Work 4: Word Count MapReduce

Distributed Systems - DS2026

Nguyen Viet Hung
Student ID: 23BI14188
Department: ICT

December 12, 2025

Contents

1 Introduction

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. The core concept involves two steps:

- **Map:** Performs filtering and sorting (e.g., sorting students by first name into queues, one queue for each name).
- **Reduce:** Performs a summary operation (e.g., counting the number of students in each queue, yielding name frequencies).

In this practical work, we aim to implement the classic "Word Count" problem. Following the course requirement to "invent yourself" a framework for C/C++, we have designed and built a custom, in-memory MapReduce engine in C++.

2 Implementation Choice

We decided to build a **Custom C++ MapReduce Simulation** instead of using existing heavyweight frameworks (like Hadoop streaming or simple-mapreduce libraries).

2.1 Reasons for Choice

1. **Educational Value:** Building the engine from scratch helps in understanding the internal mechanics of the Shuffle and Sort phases, which are often hidden in high-level frameworks.
2. **Performance:** C++ provides low-level memory management and high performance, suitable for processing text data efficiently.
3. **Simplicity:** For a single-node simulation, a custom C++ program avoids the configuration overhead of Java-based Hadoop clusters.

3 System Design

Our system mimics the standard MapReduce data flow but runs within a single process. It is divided into two layers:

- **The Framework (Engine):** Handles file I/O, data orchestration, shuffling, and sorting.
- **The User Applications:** The specific `map()` and `reduce()` functions defined by the user.

3.1 Architecture Diagram

The data flows sequentially through the phases as shown below:

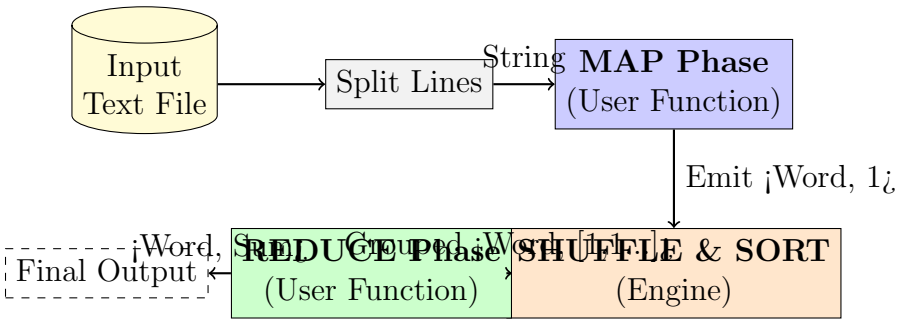

Figure 1: Custom MapReduce Data Flow Architecture

4 Implementation Details

4.1 Data Structures

We define a generic structure to hold key-value pairs, which is the fundamental unit of data exchange in MapReduce.

```
struct KeyValue {
    std::string key;
    int value;
};
```

4.2 The Mapper

The `map_func` takes a line of text, tokenizes it into words, and emits a key-value pair `<word, 1>` for every occurrence.

```
std::vector<KeyValue> map_func(const std::string& text) {
    std::vector<KeyValue> emitted_pairs;
    std::stringstream ss(text);
    std::string word;
    while (ss >> word) {
        // Optional: Cleaning punctuation can be done here
        emitted_pairs.push_back({word, 1});
    }
    return emitted_pairs;
}
```

4.3 The Shuffle & Sort Phase

This is the most critical part handled by the "main" function. We utilize the C++ STL container `std::map<std::string, std::vector<int>>`.

- **Sorting:** `std::map` automatically keeps keys sorted alphabetically (a requirement for MapReduce).
- **Grouping:** It maps a key to a *list* of values (`vector<int>`), effectively simulating the shuffle phase where all values for the same key are brought together.

4.4 The Reducer

The reducer is simple: it iterates through the list of values (which are all 1s in Word Count) and sums them up.

```cpp
int reduce_func(const std::string& key, const std::vector<int>& values) {
    int sum = 0;
    for (int v : values) {
        sum += v;
    }
    return sum;
}
```

5 Execution Results

The program is compiled using g++ via a Makefile.

```
$ make
$ ./wordcount input.txt
```

Given the sample input:

three swiss witch-bitches, which wished to be switched swiss...

The system successfully counts the frequencies:

```
[Framework] Starting MAP phase...
[Framework] Starting SHUFFLE phase...
[Framework] Starting REDUCE phase...

--- FINAL OUTPUT ---
Cause: 1
'Til: 6
'til: 1
('til: 3
And: 3
Birds: 2
But: 3
Can't: 2
Don't: 1
How: 1
I: 26
I'd: 2
I'll: 1
I'm: 6
If: 1
It: 1
Might: 1
Nothing: 1
Say: 2
Tell: 1
You: 3
a: 3
act: 1
alone: 2
alright: 1
always: 1
and: 1
```

```
another: 1
away,: 1
baby: 1
baby,: 2
be: 5
believe: 1
better: 4
bit,: 1
blue,: 1
buried: 1
but: 2
carry: 1
casket: 1
change: 2
compliments: 1
could: 2
cryin': 2
day: 5
dead: 1
die: 3
die): 2
don't: 8
even: 2
eyes: 2
eyes): 1
feather,: 2
for: 2
forever: 2
forever,: 2
full: 1
go,: 1
going: 1
goodbye: 1
grave: 1
had: 1
hm: 2
if: 4
in: 4
it: 1
it's: 5
it,: 1
keep: 1
knew: 1
know: 4
leaves: 2
left: 1
life: 1
light: 2
long,: 2
look: 2
```

```
lose: 1
love: 4
me: 2
me,: 1
might: 3
mind's: 1
more: 2
my: 3
never: 2
no: 1
not: 4
of: 3
please: 1
polluted: 1
quit,: 1
rot: 1
said: 2
same: 1
save: 1
say: 1
see: 1
see,: 1
shit,: 1
should: 2
so: 2
stay: 1
stick: 2
stupid: 1
surprised: 1
that: 5
the: 12
think: 4
throw: 1
to: 4
together,: 2
told: 1
too,: 1
turnin': 1
uh: 2
wanna: 2
want: 2
was: 1
wasn't: 2
we: 2
weather,: 2
what: 2
without: 1
would: 1
wouldn't: 1
ya: 2
```

```
you: 11
you’re: 1
you,: 2
your: 2
...
```

6 Task Distribution

- **Nguyen Viet Hung (23BI14188):**
  - Conceptualized the simulation architecture.
  - Implemented the core engine in C++ (File I/O, Shuffle logic).
  - Implemented the Map and Reduce logic for Word Count.
  - Wrote this technical report.