Practical Work 5: The Longest Path

Pham Hoang Viet
ID: 23BI14452

1 Goal

The objective of this practical work is to implement a Longest Path toy project using a MapReduce framework. The input consists of a set of files, each corresponding to one machine, where each line contains a full path to a file (as produced by the command `find /`). The output of the program is the path or paths that have the greatest length.

2 Choice of MapReduce Implementation

The implementation is written in C++17 and uses a self-made MapReduce runtime, based on the following components:

- **Threads**: `std::thread` is used to execute mappers and reducers in parallel.
- **Data structures**: intermediate key–value pairs are stored in `std::vector<std::pair<int,std::st` where the key is the length of the path and the value is the path string.
- **Partitioning**: `std::hash<int>` is used to assign each length to a specific reducer.

This approach satisfies the requirement to "invent" a simple MapReduce-style framework for C/C++.

3 Design of the Map and Reduce Functions

Map Function

The input to the map phase is the global list of paths gathered from all input files. The mapper function processes a contiguous subset of this list. For each path it computes:

- the integer length of the path (number of characters);
- a pair $(length, path)$ as intermediate output.

Each mapper produces a local vector of such pairs.

Shuffle and Partitioning

A shuffle step distributes intermediate pairs among reducers. Each pair is assigned to a reducer index computed as:

$$reducer\_id = hash(length) \bmod num\_reducers.$$

This ensures that all paths with the same length are handled by the same reducer, so that local maxima can be determined independently.

Reduce Function

Each reducer receives a list of $(length, path)$ pairs and computes:

- the maximum length seen by this reducer;
- the set of paths that achieve this maximum length.

The global maximum and final longest path set are obtained by merging the outputs of all reducers in the main thread.

4 System Organization

The system consists of one executable `longestpath` built from `longestpath.cpp`. The program is run as:

```
./longestpath output.txt paths1.txt paths2.txt ...
```

- All input files are read line by line, and all paths are stored in a vector.
- The list of paths is divided into chunks of (approximately) equal size, one chunk per mapper thread.
- Intermediate results are redistributed into reducer input lists according to the hash of the length.
- Reducers compute local longest paths; the main thread merges them to obtain the global result.

5 Implementation of the Longest Path MapReduce

Mapper and Reducer Code

System Organization

Input files $\rightarrow$ global list of paths.
Paths list $\rightarrow$ split into chunks for mapper threads.
Mappers emit $(length, path)$ pairs.
Shuffle groups pairs by length and assigns them to reducers.
Reducers compute local maxima; main thread merges them to obtain the longest path length and its corresponding paths.

Figure 1: Organization of the MapReduce longest path system.

Listing 1: Mapper and reducer workers.

```
struct ReducerResult {
    int max_len = 0;
    std::vector<std::string> paths;
};

void mapper_worker(const std::vector<std::string>& all_paths,
                   size_t begin, size_t end,
                   std::vector<std::pair<int, std::string>>& out_pairs) {
    for (size_t i = begin; i < end; ++i) {
        const std::string& p = all_paths[i];
        int len = static_cast<int>(p.size());
        out_pairs.emplace_back(len, p);
    }
}

void reducer_worker(const std::vector<std::pair<int, std::string>>&
   in_pairs,
                    ReducerResult& out_result) {
    int local_max = 0;
    std::vector<std::string> local_paths;
    for (const auto& kv : in_pairs) {
        int len = kv.first;
        const std::string& path = kv.second;
        if (len > local_max) {
            local_max = len;
            local_paths.clear();
            local_paths.push_back(path);
        } else if (len == local_max) {
            local_paths.push_back(path);
        }
    }
    out_result.max_len = local_max;
    out_result.paths = std::move(local_paths);
}
```

Main Control Flow

Listing 2: Main MapReduce control flow.

```
int main(int argc, char** argv) {
    std::string output_file = argv[1];
    std::vector<std::string> input_files;
    for (int i = 2; i < argc; ++i) input_files.push_back(argv[i]);

    std::vector<std::string> all_paths;
    for (const auto& fname : input_files) {
        std::ifstream in(fname);
        std::string line;
        while (std::getline(in, line)) {
            if (!line.empty()) all_paths.push_back(line);
        }
    }

    int num_mappers = 4;
    int num_reducers = 2;

    std::vector<std::thread> mapper_threads;
    std::vector<std::vector<std::pair<int, std::string>>> mapper_outputs(
       num_mappers);

    size_t total = all_paths.size();
    size_t base_chunk = total / num_mappers;
    size_t extra = total % num_mappers;
    size_t start = 0;

    for (int i = 0; i < num_mappers; ++i) {
        size_t chunk = base_chunk + (i < static_cast<int>(extra) ? 1 : 0);
        size_t end = start + chunk;
        mapper_threads.emplace_back(mapper_worker,
                                    std::cref(all_paths),
                                    start, end,
                                    std::ref(mapper_outputs[i]));
        start = end;
    }

    for (auto& t : mapper_threads) t.join();

    std::vector<std::vector<std::pair<int, std::string>>> reducer_inputs(
       num_reducers);
    for (const auto& m_out : mapper_outputs) {
        for (const auto& kv : m_out) {
            int len = kv.first;
            size_t h = std::hash<int>{}(len);
            int rid = static_cast<int>(h % num_reducers);
            reducer_inputs[rid].push_back(kv);
        }
    }

    std::vector<std::thread> reducer_threads;
    std::vector<ReducerResult> reducer_results(num_reducers);

    for (int r = 0; r < num_reducers; ++r) {
```

```
        reducer_threads.emplace_back(reducer_worker,
                                     std::cref(reducer_inputs[r]),
                                     std::ref(reducer_results[r]));
    }

    for (auto& t : reducer_threads) t.join();

    int global_max = 0;
    std::vector<std::string> longest_paths;
    for (const auto& res : reducer_results) {
        if (res.max_len > global_max) {
            global_max = res.max_len;
            longest_paths = res.paths;
        } else if (res.max_len == global_max) {
            longest_paths.insert(longest_paths.end(),
                                 res.paths.begin(), res.paths.end());
        }
    }

    std::ofstream out(output_file);
    for (const auto& p : longest_paths) {
        if (static_cast<int>(p.size()) == global_max)
            out << global_max << "␣" << p << "\n";
    }
    out.close();
}
```

6 Conclusion

This practical work implements a Longest Path toy project using a simple MapReduce framework in C++. The system reads multiple files containing file paths, distributes the work across mapper and reducer threads, and outputs all paths with maximum length.

The design demonstrates how MapReduce concepts such as map, shuffle, and reduce can be applied to non-text-processing problems and provides a basis for more advanced distributed algorithms.