TK1 Theory Exercise 7

Group 21
Li Kecen
Liu Shule
Feng Letian
Chen Zhen
Zhang Xin

Task1

1. Input Reader

Read input data from storage as a form of key value pairs for later processing.

2. Map

Map the input key value pairs into an intermediate result, also in the form of key value pairs, which will facilitate the reduction later on to get the final result.

3. Partition

Map worker will distribute the intermediate key value pairs to reducer. All pairs with the same key will be distributed to a single reducer, but a reducer might be assigned pairs of more than one key.

4. Compare

Compare the key value pairs assigned to a reducer and group those with the same key which shall be reduced together.

5. Reduce

Apply reduce function to groups of key value pairs, each group with the same key, to get the final result value.

6. Output Writer

Record the final result into the system.

Task2

```
map(String input_key, String input_value) {
        // input_key: name of the log file
        // input_value: the content of the log file
        for each line line_of_log in input_value :
                String url = line_of_log.substring(0, line_of_log.indexOf(" ")); // getting the url of the page visited
                EmitIntermediate(url, "1");
}
```

```
partition(int numOfReducer, Pair<String, String> intermediate) {
        // numOfReducer: number of reducers in the system
        // intermediate: intermediate result key value pair received from mapper
        int reducerIndex = hash(intermediate.getKey()) % numOfReducer;
        SendToReducer(reducerIndex);
}
```

```
Compare(Iterator intermediate_pairs, Map<String, List<intermediate_pair>> intermediate_groups) {
        // intermediate_pairs: all key value pairs received, with different keys
        // intermediate_groups: HashMap that store all key value pairs in groups based on their key
        for each pair intermediate_pair in intermediate_pairs :
                if (!intermediate_groups.containsKey(intermediate_pair.getKey())) :
                        intermediate_groups.put(intermediate_pair.getKey(), new
List<intermediate_pair>());
                intermediate_groups.get(intermediate_pair.getKey()).add(intermediate_pair);
}

reduce(String output_key, Iterator intermediate_values) {
        // output_key: url of the page visited, key of the hashmap intermediate_groups from above
        // output_values: aggregated visit count for the specific url identified by output_key
        int count = 0;
        for each v in intermediate_values :
                count += ParseInt(v);
        Emit(AsString(count));
}

output(Map<String, String> count_list) {
        // count_list: result count of each url (url: count)
        For each v in count_list :
                writeToFile(v);
}
```

Task3

a)
A mobile agent is a self-controlled process that can move within a network from one computer to another and can continue executing tasks without losing states.

b)
1. Mobile agent freeze its execution at the current host
2. Mobile agent record its states regarding agent instance variables, call stack, instruction pointers, code and probably context informations and send them to the target computer.
3. At the destination computer, mobile agent resume its execution based on the states it record and sent previously.