Verteilte Systeme

Übung 4

Philipp Borgers, Maximilian Michels, Sascha S.

main.cpp:

```
#include <mpi.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include <stdlib.h> //malloc

#include "map.hpp"

//stolen from http://www.cse.yorku.ca/~oz/hash.html
//generate a hash (unsigned long) from a string
//used to assign words to processes
unsigned long hash(std::string s) {
    unsigned long hash = 5381;

    for(std::string::const_iterator iter = s.begin(); iter != s.end(); ++iter) {
        hash = ((hash << 5) + hash) + *iter; /* hash * 33 + c */
    }
    return hash;
}

//merge a map into another map
void mergemaps(std::map<std::string, int>& result, std::map<std::string, int> map) {

    std::map<std::string, int>::iterator end = map.end();
    for(std::map<std::string, int>::iterator iter = map.begin(); iter != end; ++iter) {
        result[iter->first] = result[iter->first] + iter->second;
    }
}

int main(int argc, char* argv[]) {

    MPI_Init(&argc, &argv);

    int numprocs;
    int processid;

    //get number of process and process id
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &processid);

    std::vector<std::string> files;

    int i = 1 + processid;

    //each process should pick his files
    //we don't need a fancy distribution function so far
    for(i ; i < argc; i = i + numprocs) {
        files.push_back(std::string(argv[i]));
    }

    //container for results from the local map step
    std::map<std::string, int> localresult;

    //iterate over all files
    //collect results in vector
    std::vector<std::string>::iterator end = files.end();
```

```
    //map step
    //do map and merge local map results
    for(std::vector<std::string>::iterator iter = files.begin(); iter != end; ++iter) {
        mergemaps(localresult, mymap(*iter));
    }

    //initialize array for data that belongs to remote processes
    std::vector<std::map<std::string, int> > remoteresults;
    for(i = 0; i < numprocs; ++i) {
        remoteresults.push_back(std::map<std::string, int>());
    }

    //for each word in the local result set
    //hash the word with the hash function
    //assign word to process
    //use modulo process number
    //should garantee a even distribution over all processes
    std::map<std::string, int>::iterator end2 = localresult.end();
    for(std::map<std::string, int>::iterator iter = localresult.begin(); iter != end2; +
+iter) {
        int processid = (int) hash(iter->first) % numprocs;
        remoteresults[processid][iter->first] = iter->second;
    }

    //send local results that don't belong to process to n-1 other processes

    std::map<std::string, int> reduceresults;

    for(i = 0; i < numprocs; ++i) {
        if(i == processid) {
            //do not send data to yourself
            //merge with reduce results ;)
            mergemaps(reduceresults, remoteresults[i]);
        } else {
            //MPI_send to n-1 process
            //send a string that represents the map for process x
            std::string payload = mapserialize(remoteresults[i]);
            MPI_Send((void *)(payload.c_str()), strlen(payload.c_str())+1, MPI_CHAR, i,
1, MPI_COMM_WORLD);
        }
    }

    //recv from n-1 process
    for(i = 0; i < numprocs; ++i) {
        if(i != processid) {

            //TODO this is insane shit... simplify this!
            int count;
            MPI_Status status;
            MPI_Probe(i, 1, MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_CHAR, &count);
            char* buffer = (char*) malloc(sizeof(char) * count); //is this c++ style?

            MPI_Recv(buffer, count, MPI_CHAR, i, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

            //do not merge empty data
            if(std::string(buffer) != "") {
                //reduce data from n-1 processes into one map
                mergemaps(reduceresults, mapdeserialize(std::string(buffer)));
            }
        }
    }

    //write local data to file
    //end of reduce step
    std::stringstream ss;
    ss << "./data/reduced-" << processid;
```

```
    std::ofstream reducefile(ss.str().c_str());

    std::map<std::string, int>::iterator end4 = reduceresults.end();
    for(std::map<std::string, int>::iterator iter = reduceresults.begin(); iter != end4;
++iter) {
        reducefile << iter->first << ":" << iter->second << std::endl;
    }

    reducefile.close();

    MPI_Finalize();

    return 0;
}
```

map.hpp

```
std::map<std::string, int> mymap(std::string filename);
void printmap(std::map<std::string, int>);

std::map<std::string, int> mapdeserialize(std::string serializedmap);
std::string mapserialize(std::map<std::string, int> map);
```

map.cpp:

```
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <map>

#include "map.hpp"

//stolen from stackoverflow
void replaceAll(std::string& str, const std::string& from, const std::string& to) {
    size_t start_pos = 0;
    while((start_pos = str.find(from, start_pos)) != std::string::npos) {
        str.replace(start_pos, from.length(), to);
        start_pos += to.length(); // In case 'to' contains 'from', like replacing 'x'
with 'yx'
    }
}

//do a word count on a local file
std::map<std::string, int> mymap(std::string filename) {

    std::string word;
    std::string line;

    std::map<std::string, int> wordcounts;

    std::ifstream input(filename.c_str());

    if(input.is_open()) {

        while(input.good()) {

            //read a line from input file
            getline(input, line);

            //handle special characters as whitespace
            replaceAll(line, ".", " ");
            replaceAll(line, ",", " ");
            replaceAll(line, ";", " ");
            replaceAll(line, ":", " ");
            replaceAll(line, "-", " ");
            replaceAll(line, "+", " ");
```

```
            replaceAll(line, "/", " ");
            replaceAll(line, "?", " ");
            replaceAll(line, "!", " ");
            replaceAll(line, "(", " ");
            replaceAll(line, ")", " ");
            replaceAll(line, "[", " ");
            replaceAll(line, "]", " ");
            //replace numbers
            replaceAll(line, "0", " ");
            replaceAll(line, "1", " ");
            replaceAll(line, "2", " ");
            replaceAll(line, "3", " ");
            replaceAll(line, "4", " ");
            replaceAll(line, "5", " ");
            replaceAll(line, "6", " ");
            replaceAll(line, "7", " ");
            replaceAll(line, "8", " ");
            replaceAll(line, "9", " ");

            //be case insensetive
            std::transform(line.begin(), line.end(), line.begin(), ::tolower);

            std::stringstream linestream(line);

            //get words, split by whitespaces
            while(getline(linestream, word, ' ')) {
                if(word != "") {
                    ++wordcounts[word];
                }
            }
        }
    } else {
        //TODO some error handling?
    }

    return wordcounts;
}

//serialize a map with words
//seperate word and count by ":"
//seperate tuples by white space
std::string mapserialize(std::map<std::string, int> map) {

    std::stringstream result;

    std::map<std::string, int>::iterator end = map.end();
    for(std::map<std::string, int>::iterator iter = map.begin(); iter != end; ++iter) {
        result << iter->first << ":" << iter->second << " ";
    }

    return result.str();
}

//function to deserialize a map given as string
std::map<std::string, int> mapdeserialize(std::string serializedmap) {

    std::string tuple;
    std::map<std::string, int> map;
    std::stringstream input(serializedmap);

    std::string word;
    int count;
    size_t position = 0;

    //tokenize by whitespace
    while(getline(input, tuple, ' ')) {

        //find key-value delimiter
        //reconstruct word and wordcount
```

```
        //TODO maybe use strtok instead
        position = tuple.find(":");
        word = tuple.substr(0,position); //get word
        count = atoi(tuple.substr(position+1, tuple.length()).c_str()); //get word count
        map[word] = count; //feed data into result map
    }

    return map;
}

//debug function to print a map
void printmap(std::map<std::string, int> map) {
    std::map<std::string, int>::iterator end = map.end();
    for(std::map<std::string, int>::iterator iter = map.begin(); iter != end; ++iter) {
        std::cout << iter->first << ":" << iter->second << std::endl;
    }
}

#ifdef DEBUG
int main(int argc, char* argv[]) {
    //test of map function
    printmap(mymap("sample.txt"));
    std::cout << "########################" << std::endl;
    std::cout << mapserialize(mymap("sample.txt")) << std::endl;
    std::cout << "########################" << std::endl;
    printmap(mapdeserialize(mapserialize(mymap("sample.txt"))));

    return 0;
}
#endif
```

Testläufe

```
>make compile
>make run
mpirun -n 2 main sample.txt sample.txt sample.txt
ls -lh data
insgesamt 8,0K
-rw-r--r-- 1 mxm students  7 24. Mai 13:41 reduced-0
-rw-r--r-- 1 mxm students 47 24. Mai 13:41 reduced-1
cat data/*
dies:9
ein:9
ist:9
mit:3
ohne:6
punkt:6
satz:9
sinn:3

>make clean
>make run8
mpirun -n 2 main sample.txt sample.txt sample.txt sample.txt sample.txt sample.txt
sample.txt sample.txt
ls -lh data
insgesamt 8,0K
-rw-r--r-- 1 mxm students  8 24. Mai 13:48 reduced-0
-rw-r--r-- 1 mxm students 52 24. Mai 13:48 reduced-1
cat data/*
dies:24
ein:24
ist:24
mit:8
ohne:16
punkt:16
satz:24
sinn:8
```

Dokumentation

Die Eingabe für den Map/Reduce-Process sind die Dateinamen der Dateien, deren Word-Count berechnet werden soll. Die Daten sollen im Reduce-Schritt auf den verschiedenen Prozessen zusammengeführt werden.

Allen Prozessen stehen durch MPI die Eingabeparameter zur Verfügung. Jeder Prozess wählt sich aus den vorhandenen Dateinahmen anhand seiner Prozess-Id Dateien für seinen Map-Schritt aus. Die Dateien werden der Reihe nach auf die Prozesse verteilt. Wir beginnen bei dem Prozess mit der Id 0 und verteilen aufsteigend weiter. Gibt es mehr Dateien als Prozesse, werden die überschüssigen Dateien wieder bei Prozess 0 beginnend weiterverteilt (Modulo Anzahl Prozesse).

Jeder Prozess zählt im Map-Schritt die Anzahl der Wörter in seinen Dateien und führt die Ergebnisse lokal zusammen.

Im Reduce-Schritt werden die Daten auf die Combiner verteilt. Jeder Prozess ermittelt per Hash-Funktion welcher Prozess für welches seiner Wörter verantwortlich ist. Wurden alle Daten aufgeteilt, sendet jeder Prozess die Daten an die entsprechenden Threads. Alle vorhanden Prozesse nehmen am Combining teil. Auch Prozesse, die keine Wörter gezählt haben. Insgesamt werden n * n - 1 vieleErgebnisse verschickt unter den Prozessen.

Sobald ein Prozess alle Daten erhalten hat und sie mit seinen lokalen Daten zusammengeführt hat, werden die Daten auf die Festplatte geschrieben und der Reduce-Schritt wird damit beendet.

Unklar ist, warum für die Zuordnung der Dateinamen zu Prozessen eine Hash-Funktion verwendet werden sollte. Gehen wir davon aus, dass ein Word-Count nur einmal ausgeführt wird, ist es unerheblich, ob wir eine Hash-Funktion benutzen oder nicht. Werden mehrere Word-Counts gleichzeitig ausgeführt, könnte ein Hashing der Dateinamen für eine gleichmäßigere Auslastung der einzelnen Prozesse sorgen. Die Größe der Dateien ist jedoch der eigentliche Faktor für die Berechnungszeit. Dieses Problem wird durch das Hashen der Dateinamen nicht gelöst.