Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«Распределённые системы»

Разработка алгоритмов надёжного хранения и обработки данных

ОТЧЕТ

о выполненном задании

студента 427 учебной группы факультета ВМК МГУ
Галустова Артемия Львовича

гор. Москва
2022 год

Содержание

Протокол голосования

Постановка задачи

Рассматривается система состоящая из клиента и N файловых серверов на которых хранятся размноженные копии файлов. Требуется реализовать протокол голосования при размножении файлов. Также требуется дать оценку времени выполнения данного алгоритма.

Описание алгоритма

Сущность метода голосования при размножении заключается в выполнении запросе чтения и записи у многих серверов.

Для N серверов выбираются два числа N_r, N_w - кворумы на чтение и запись, соответственно, такие что $N_r + N_w > N$. При выполнении операции чтения клиент должен обратиться к N_r серверам. Каждый возвращает содержимое файла и версию. Клиент выбирает файл с максимальной версией. При выполнении операции записи требуется установить согласие между серверами относительно следующей версии файла, после чего запросить запись у всех серверов с новой версией и считать успешной, если как минимум N_w серверов выполнили её.

Оценка времени выполнения

Требуется дать оценку времени выполнения одним процессом 2-х операций записи и 10 операций чтения $L = 300$ байтов информации с файлом, расположенным на остальных $N = 10$ ЭВМ сети с шинной организацией (без аппаратных возможностей широковещания). Время старта и время передачи одного байта принимаются равными $T_s = 100$ и $T_b = 1$ соответственно.

Дополнительно предположим, что версия файла и имя (индекс) файла на сервере занимают по $R = 8$ байт, а тип запроса не требуется отдельно кодировать. Для чтения файла необходимо запросить у N_r серверов данные. Каждому серверу будет отправлен запрос с именем файла и в ответ получен файл и его версия. Итого для каждого запроса на чтение $T_r = N_r \cdot ((T_s + T_b R) + (T_s + T_b(R + L)))$. Для чтения сперва необходимо установить новую версию файла, запросив старую у N_r серверов, передав им имя файла и получив версию в ответ. После чего передать *всем* серверам новый файл, его имя и новую версию и получить ответ от N_w серверов (а остальные проигнорировать). Итого для каждого запроса на запись $T_w = N_r \cdot ((T_s + T_b R) + (T_s + T_b R)) + N \cdot (T_s + T_b(R + L)) + N_w T_s$. Всего запросы будут выполняться $T = 2T_w + 10T_r$. Наложим условие $N_r + N_w > N$ и произведём оптимизацию по N_r и N_w в $\mathbb{N}$. Как и ожидается минимум достигается при $N_r = 1$, $N_w = 10$. При этом время работы составит $T = 15752$.

Реализация

Алгоритм был реализован на языке C++ с использованием библиотеки передачи сообщений MPI. Код реализации доступен в приложенном архиве, по адресу `https://github.com/NotLebedev/dist_task/tree/master/voting-protocol`,

а также частично в «Исходный код для протокола голосования». Была реализована модельная система из одного клиента и нескольких серверов, обменивающихся сообщениями через систему MPI (см. файлы Client.h и Server.h).

Для запуска необходимо собрать проект при помощи утилиты `CMake` командой `cmake - -build ./cmake-build-debug - -target integrals_ulfm`. После чего полученный файл `./cmake-build-debug/integrals_ulfm` запускается с единственной опцией - именем конфигурационного файла (вместе с исходным кодом поставляются два файла - `faulty_reads.toml`, `faulty_writes.toml`, демонстрирующие обработку неуспешного чтения и записи). При помощи переменной среды `READ_QUORUM` можно выставить кворум на чтение (кворум на запись автоматически выбирается как $N - N_r + 1$. После запуска программа выведет отчёт о выполнении операций заданных в файле конфигурации и произошедших ошибках.

Результаты

Ниже приведён вывод для конфигураций `faulty_reads.toml` и `faulty_writes.toml` при `READ_QUORUM` равном 3 на 8 процессах (1 клиент, 7 серверов)

faulty_reads

```
Read quorum is: 3
Write quorum is: 5
Started client. Reading configuration.
Files distributed before start:
        blindsight = "It didn't start out here. Not with the scramblers or
            ↪ Rorschach, not with Big Ben or Theseus or the vampires."
        dune = "A beginning is the time for taking the most delicate care that
            ↪ the balances are correct."
        farehnheit451 = "It was a pleasure to burn. It was a special pleasure to
            ↪ see things eaten, to see things blackened and changed."
Operations to be performed:
        Read file blindsight
        Disable server 1
        Read file blindsight
        Disable server 2
        Read file blindsight
        Disable server 3
        Read file blindsight
        Disable server 4
        Read file blindsight
        Disable server 5
        Read file blindsight

Next command:
        Read file blindsight
```

Read successfull. Got versions 1 1 1 Latest versions had contents:
"It didn't start out here. Not with the scramblers or Rorschach, not with Big Ben or Theseus or the vampires."

Next command:
Disable server 1

Next command:
Read file blindsight
Read successfull. Got versions 1 1 1 Latest versions had contents:
"It didn't start out here. Not with the scramblers or Rorschach, not with Big Ben or Theseus or the vampires."

Next command:
Disable server 2

Next command:
Read file blindsight
Read successfull. Got versions 1 1 1 Latest versions had contents:
"It didn't start out here. Not with the scramblers or Rorschach, not with Big Ben or Theseus or the vampires."

Next command:
Disable server 3

Next command:
Read file blindsight
Read successfull. Got versions 1 1 1 Latest versions had contents:
"It didn't start out here. Not with the scramblers or Rorschach, not with Big Ben or Theseus or the vampires."

Next command:
Disable server 4

Next command:
Read file blindsight
Read successfull. Got versions 1 1 1 Latest versions had contents:

```
"It didn't start out here. Not with the scramblers or Rorschach, not with Big
    ↪ Ben or Theseus or the vampires."


Next command:
        Disable server 5


Next command:
        Read file blindsight
Failed to read. Quorum not met



Process finished with exit code 0
```

faulty_writes

```
Read quorum is: 3
Write quorum is: 5
Started client. Reading configuration.
Files distributed before start:
        blindsight = "It didn't start out here. Not with the scramblers or
            ↪ Rorschach, not with Big Ben or Theseus or the vampires."
        dune = "A beginning is the time for taking the most delicate care that
            ↪ the balances are correct."
        farehnheit451 = "It was a pleasure to burn. It was a special pleasure to
            ↪ see things eaten, to see things blackened and changed."
Operations to be performed:
        Read file dune
        Write contents "This every sister of the Bene Gesserit knows." to file dune
        Read file dune
        Read file blindsight
        Disable server 1
        Disable server 2
        Disable server 3
        Write contents "To begin your study of the life of 'MuadDib, then, take
            ↪ care that you first place him in his time: born in the 57th year of
            ↪ the Padishah Emperor, Shaddam IV." to file dune
        Enable server 1
        Enable server 2
        Enable server 3
        Read file dune
        Disable server 1
        Disable server 3
```

```
        Write contents "To begin your study of the life of ‘MuadDib, then, take
            ↪ care that you first place him in his time: born in the 57th year of
            ↪ the Padishah Emperor, Shaddam IV." to file dune
        Enable server 1
        Enable server 3
        Read file dune

Next command:
        Read file dune
Read successfull. Got versions 1 1 1 Latest versions had contents:
 "A beginning is the time for taking the most delicate care that the balances are
     ↪ correct."


Next command:
        Write contents "This every sister of the Bene Gesserit knows." to file dune
Successful write. Updated files to version 2


Next command:
        Read file dune
Read successfull. Got versions 2 2 2 Latest versions had contents:
 "This every sister of the Bene Gesserit knows."


Next command:
        Read file blindsight
Read successfull. Got versions 1 1 1 Latest versions had contents:
 "It didn’t start out here. Not with the scramblers or Rorschach, not with Big
     ↪ Ben or Theseus or the vampires."


Next command:
        Disable server 1


Next command:
        Disable server 2


Next command:
        Disable server 3


Next command:
```

```
        Write contents "To begin your study of the life of ‘MuadDib, then, take
            ↪ care that you first place him in his time: born in the 57th year of
            ↪ the Padishah Emperor, Shaddam IV." to file dune
Failed to write. Quorum not met


Next command:
        Enable server 1


Next command:
        Enable server 2


Next command:
        Enable server 3


Next command:
        Read file dune
Read successfull. Got versions 2 2 2 Latest versions had contents:
 "This every sister of the Bene Gesserit knows."


Next command:
        Disable server 1


Next command:
        Disable server 3


Next command:
        Write contents "To begin your study of the life of ‘MuadDib, then, take
            ↪ care that you first place him in his time: born in the 57th year of
            ↪ the Padishah Emperor, Shaddam IV." to file dune
Successful write. Updated files to version 3


Next command:
        Enable server 1


Next command:
        Enable server 3
```

```
Next command:
        Read file dune
Read successfull. Got versions 2 3 2 Latest versions had contents:
 "To begin your study of the life of ‘MuadDib, then, take care that you first place
    ↪ him in his time: born in the 57th year of the Padishah Emperor,
    ↪ Shaddam IV."



Process finished with exit code 0
```

Выводы

Удалось реализовать протокол голосования при работе с размноженными файлами и продемонстрировать его работу на модельной системе. Протокол корректно обрабатывает ситуации отказов файловых серверов, как в случае достаточного так и в случае недостаточного кворума.

Отказоустойчивая MPI программа

Постановка задачи

Рассматривается программа, рассчитывающая определённый интеграл реализованная в рамках курса "Суперкомпьютеры и параллельная обработка данных"(см. `https://github.com/NotLebedev/integralsOpenMP/tree/mpi-impl`). Требуется добавить контрольные точки для продолжения работы программы в случае сбоя. Реализовать один из 3-х сценариев работы после сбоя: а) продолжить работу программы только на “исправных” процессах; б) вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов; в) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

Описание алгоритма

Детали реализации оригинального алгоритма доступны в старом отчёте по ссылке указанной выше. Для реализации отказоустойчивой версии важен порядок работы алгоритма: мастер-процесс (номер 0) разделяет интервал на котором осуществляется интегрирование на N - число потоков, частей. После чего каждому потоку отправляется информация о его интервале (задание), производится вычисление заданий и результат редуцируется. Для реализации отказоустойчивости была выбран сценарий продолжения работы на "исправных процессах". Мастер процесс создаёт начальную контрольную точку: запоминает все задания, текущую сумму (в начале она 0); и запускает вычисление. Если после завершения вычислений был зафиксирован отказ некоторых процессов, то из контрольной точки восстанавливаются их задания, создаётся новая контрольная точка с оставшимися заданиями, промежуточной суммой (от успешно завершившихся процессов); и оставшиеся задания рассылаются оставшимся процессам. Это повторяется пока все задания не будут выполнены.

Реализация

Код реализации доступен в приложенном архиве, по адресу `https://github.com/NotLebedev/dist_task/tree/master/integrals-ulfm`, а также частично в «Исходный код для отказоустойчивой MPI программы».

Для запуска, как и для первого проекта необходимо собрать проект при помощи утилиты `CMake`. Команда `cmake - -build ./cmake-build-debug - -target voting_protocol`. После чего полученный файл `./cmake-build-debug/voting_protocol` запускается с двумя опциями - количеством шагов алгоритма интегрирования (позволяет регулировать время выполнения) и количеством процессов, которые откажут во время выполнения.

Результаты

Ниже приведены результаты запуска без отказов, с одним отказом и отказом всех процессов кроме трёх. Тестирование выполнялось на 12 процессах

Без отказов

```
Unfinished jobs {}
All done!
Runtime: 4.95214 seconds. Result: 3.3504

Process finished with exit code 0
```

Один отказ

```
Rank 10 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 8 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 3 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 6 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 2 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 4 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 0 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 5 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 7 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 9 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Rank 1 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 1
    ↪ found dead: { 11 }
Unfinished jobs {11, }
Unfinished jobs {}
All done!
Runtime: 8.26728 seconds. Result: 3.3504

Process finished with exit code 0
```

Отказ всех кроме трёх

```
Rank 0 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 9
    ↪ found dead: { 3 4 5 6 7 8 9 10 11 }
Rank 1 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 9
    ↪ found dead: { 3 4 5 6 7 8 9 10 11 }
Rank 2 / 12: Notified of error MPI_ERR_PROC_FAILED: Process Failure. 9
    ↪ found dead: { 3 4 5 6 7 8 9 10 11 }
Unfinished jobs {3, 4, 5, 6, 7, 8, 9, 10, 11, }
Unfinished jobs {3, 4, 5, 6, 7, 8, }
Unfinished jobs {3, 4, 5, }
Unfinished jobs {}
All done!
Runtime: 14.8369 seconds. Result: 3.3504

Process finished with exit code 0
```

Выводы

Удалось модифицировать существующую программу, чтобы добиться отказоустойчивого поведения и не изменить результаты вычисления. Программа может справиться с отказом единичных процессов, а также отказом большей части процессов.

Исходный код для протокола голосования

Файл Client.h

```
#ifndef DIST_TASK_CLIENT_H
#define DIST_TASK_CLIENT_H


#include <string>
#include <memory>
#include <vector>
#include <optional>
#include <map>
#include "Commands.h"

class Client {
public:
    explicit Client(const std::string &filename);

    void run();
protected:
    class JobSequence {
    public:
        explicit JobSequence(const std::string &filename);

        [[nodiscard]] const std::map<std::string, std::string> &getInitialFiles()
    ↪  const;

        [[nodiscard]] const std::vector<std::unique_ptr<Command>> &
    ↪ getCommandSequence() const;

    private:
        std::map<std::string, std::string> initial_files;
        std::vector<std::unique_ptr<Command>> command_sequence;
    };

private:
    ssize_t getServerCount();
    void copyFilesToServers();
    void copyFilesToOneServer(int serverIdx);
    void processCommands();
    void stopServers();

    void handleCommandWrite(Write *command);
    void handleCommandRead(Read *command);
    static void handleCommandDisableServer(DisableServer *command);
    static void handleCommandEnableServer(EnableServer *command);

    static int sendWriteMessage(int serverIdx, int nextVersion, const std::string
    ↪  &filename, const std::string &content);
    static std::optional<std::tuple<int, std::string>> sendReadMessage(int
    ↪ serverIdx, const std::string &filename);
    static int sendGetVersion(int serverIdx, const std::string &filename);
    static void sendRawCommandType(int serverIdx, CommandType commandType);
```

```
    std::unique_ptr<JobSequence> jobSequence;
    ssize_t server_count = -1;
    size_t read_quorum = 1;
    size_t write_quorum;
};


#endif //DIST_TASK_CLIENT_H
```

Файл Client.cpp

```
#include <vector>
#include <map>
#include <toml++/toml.h>
#include <iostream>
#include <mpi.h>
#include "Client.h"
#include "Commands.h"
#include "MPIHelper.h"

void Client::run() {
    std::cout << "Started client. Reading configuration.\n";
    std::cout << "Files distributed before start:\n";
    for (const auto &file: jobSequence->getInitialFiles())
        std::cout << "\t" << file.first << " = \"" << file.second << "\"\n";

    std::cout << "Operations to be performed:\n";
    for (const auto &operation: jobSequence->getCommandSequence())
        std::cout << "\t" << operation->describe() << "\n";
    std::cout << std::endl;

    copyFilesToServers();

    processCommands();

    stopServers();
}

Client::Client(const std::string &filename) : jobSequence(std::make_unique<
    ↪ JobSequence>(filename)) {
    if(const char* env_p = std::getenv("READ_QUORUM")) {
        try {
            int env_p_to_int = std::stoi(std::string(env_p));
            if (env_p_to_int > 0 && env_p_to_int <= getServerCount()) {
                read_quorum = env_p_to_int;
            } else
                std::cout << "Incorrect value in READ_QUORUM variable. Falling
    ↪ bact to defaults" << "\n";
        } catch (std::invalid_argument &e) {
            std::cout << "Incorrect value in READ_QUORUM variable. Falling bact
    ↪ to defaults" << "\n";
        }
    }
    write_quorum = getServerCount() - read_quorum + 1;
    std::cout << "Read quorum is: " << read_quorum << '\n' << "Write quorum is: "
    ↪  << write_quorum << std::endl;
}

void Client::processCommands() {
    for (const auto &command: jobSequence->getCommandSequence()) {
        std::cout << "Next command:\n\t" << command->describe() << std::endl;
        switch (command->getType()) {
            case CommandRead:
                handleCommandRead(dynamic_cast<Read *>(command.get()));
                break;
```

```
            case CommandWrite:
                handleCommandWrite(dynamic_cast<Write *>(command.get()));
                break;
            case CommandDisableServer:
                handleCommandDisableServer(dynamic_cast<DisableServer *>(command.
    ↪ get()));
                break;
            case CommandEnableServer:
                handleCommandEnableServer(dynamic_cast<EnableServer *>(command.
    ↪ get()));
                break;
            // Not available in config
            case CommandGetVersion:
            case CommandStopServer:
                break;
        }
        std::cout << "\n" << std::endl;
    }
}

ssize_t Client::getServerCount() {
    if (server_count == -1) {
        int num_process;
        MPI_Comm_size(MPI_COMM_WORLD, &num_process);

        server_count = num_process - 1;
    }

    return server_count;
}

void Client::copyFilesToServers() {
    for (int i = 0; i < this->getServerCount(); i++) {
        copyFilesToOneServer(i + 1);
    }
}

void Client::copyFilesToOneServer(int serverIdx) {
    for (const auto &file: jobSequence->getInitialFiles()) {
        sendWriteMessage(serverIdx, 1, file.first, file.second);
    }
}

int Client::sendWriteMessage(int serverIdx, int nextVersion, const std::string &
    ↪ filename, const std::string &content) {
    if (serverIdx == 0)
        return -1;

    int pos = 0;
    int message_type = CommandType::CommandWrite;

    MPIPackBufferFactory bufferFactory{};
    bufferFactory.addInt(2); // Type of message and version
    bufferFactory.addString(filename);
    bufferFactory.addString(content);
```

```
    std::vector<uint8_t> buf = bufferFactory.getBuf();

    MPI_pack_int(message_type, buf, &pos);
    MPI_pack_int(nextVersion, buf, &pos);
    MPI_pack_string(filename, buf, &pos);
    MPI_pack_string(content, buf, &pos);

    MPI_Send(buf.data(), pos, MPI_PACKED, serverIdx, 0, MPI_COMM_WORLD);

    MPI_Status status;
    int answer;
    MPI_Recv(&answer, 1, MPI_INT, serverIdx, 0, MPI_COMM_WORLD, &status);

    if (status.MPI_ERROR != MPI_SUCCESS || !answer)
        return 1;
    else
        return 0;
}

std::optional<std::tuple<int, std::string>> Client::sendReadMessage(int serverIdx
    ↪ , const std::string &filename) {
    if (serverIdx == 0)
        return {};

    int pos = 0;
    int message_type = CommandType::CommandRead;

    MPIPackBufferFactory bufferFactory{};
    bufferFactory.addInt(1); // Type of message
    bufferFactory.addString(filename);

    std::vector<uint8_t> buf = bufferFactory.getBuf();

    MPI_pack_int(message_type, buf, &pos);
    MPI_pack_string(filename, buf, &pos);

    MPI_Send(buf.data(), pos, MPI_PACKED, serverIdx, 0, MPI_COMM_WORLD);

    // Receive contents of file
    MPI_Status status;
    // Probe for message to get buffer size
    MPI_Probe(serverIdx, 0, MPI_COMM_WORLD, &status);
    // Get buffer size, before receiving
    int buf_len;
    MPI_Get_count(&status, MPI_PACKED, &buf_len);
    std::vector<uint8_t> getBuf(buf_len);
    MPI_Recv(getBuf.data(), buf_len, MPI_PACKED, serverIdx, 0, MPI_COMM_WORLD, &
    ↪ status);

    pos = 0;
    int version = MPI_unpack_int(getBuf, &pos);

    if (version < 0)
        return {};
    auto content = MPI_unpack_string(getBuf, &pos);
```

```
    return std::tuple(version, content);
}

int Client::sendGetVersion(int serverIdx, const std::string &filename) {
    if (serverIdx == 0)
        return -1;

    int pos = 0;
    int message_type = CommandType::CommandGetVersion;

    MPIPackBufferFactory bufferFactory{};
    bufferFactory.addInt(1); // Type of message
    bufferFactory.addString(filename);

    std::vector<uint8_t> buf = bufferFactory.getBuf();

    MPI_pack_int(message_type, buf, &pos);
    MPI_pack_string(filename, buf, &pos);

    MPI_Send(buf.data(), pos, MPI_PACKED, serverIdx, 0, MPI_COMM_WORLD);

    MPI_Status status;
    int version;
    MPI_Recv(&version, 1, MPI_INT, serverIdx, 0, MPI_COMM_WORLD, &status);

    return version;
}

void Client::sendRawCommandType(int serverIdx, CommandType commandType) {
    if (serverIdx == 0)
        return;

    int pos = 0;
    int message_type = commandType;

    MPIPackBufferFactory bufferFactory{};
    bufferFactory.addInt(1); // Type of message

    std::vector<uint8_t> buf = bufferFactory.getBuf();

    MPI_pack_int(message_type, buf, &pos);

    MPI_Send(buf.data(), pos, MPI_PACKED, serverIdx, 0, MPI_COMM_WORLD);
}

void Client::handleCommandWrite(Write *command) {
    const std::string &filename = command->getFilename();
    const std::string &contents = command->getContents();

    std::vector<int> versions{};
    for (int i = 0; i < getServerCount(); i++) {
        int version = sendGetVersion(i + 1, filename);
        if (version > 0)
            versions.push_back(version);
        if (versions.size() == write_quorum)
            break;
```

```
    }

    // Here it is assumed that failure to get quorum on version is same as
    ↪ failure to write
    // It is totally possible to check that all servers accepted write, but that
    ↪ will require
    // implementing rollbacks and that is quite beyond scope of this task
    if (versions.size() < write_quorum) {
        std::cout << "Failed to write. Quorum not met" << std::endl;
        return;
    }

    int new_version = *std::max_element(versions.cbegin(), versions.cend()) + 1;
    for (int i = 0; i < getServerCount(); i++) {
        sendWriteMessage(i + 1, new_version, filename, contents);
    }
    std::cout << "Successful write. Updated files to version " << new_version <<
    ↪ std::endl;
}

void Client::handleCommandRead(Read *command) {
    const std::string &filename = command->getFilename();

    std::vector<int> versions{};
    std::vector<std::string> contents{};
    for (int i = 0; i < getServerCount(); i++) {
        auto res = sendReadMessage(i + 1, filename);
        if (res.has_value()) {
            versions.push_back(get<0>(res.value()));
            contents.push_back(get<1>(res.value()));
        }
        if (versions.size() == read_quorum)
            break;
    }

    if (versions.size() < read_quorum) {
        std::cout << "Failed to read. Quorum not met" << std::endl;
        return;
    }

    auto max_index = std::distance(versions.begin(), std::max_element(versions.
    ↪ begin(), versions.end()));
    std::string &result = contents[max_index];

    std::cout << "Read successfull. Got versions ";
    for (const auto version: versions)
        std::cout << version << " ";
    std::cout << " Latest versions had contents:\n \"" << result << "\"" << std::
    ↪ endl;
}

void Client::handleCommandDisableServer(DisableServer *command) {
    sendRawCommandType(command->getServer(), CommandDisableServer);
}

void Client::stopServers() {
```

```
    for (int i = 0; i < this->getServerCount(); i++) {
        sendRawCommandType(i + 1, CommandType::CommandStopServer);
    }
}

void Client::handleCommandEnableServer(EnableServer *command) {
    sendRawCommandType(command->getServer(), CommandEnableServer);
}

Client::JobSequence::JobSequence(const std::string &filename) : initial_files(),
    ↪ command_sequence() {
    toml::table tbl;
    try {
        tbl = toml::parse_file(filename);
    } catch (const toml::parse_error &err) {
        std::cerr << "Parsing failed:\n" << err << "\n";
        throw err;
    }

    for (const auto &file: *tbl["Initial"].as_table()) {
        initial_files[std::string(file.first)] = std::string(*file.second.
    ↪ as_string());
    }

    for (const auto &command: *tbl["Command"].as_array()) {
        const toml::table *tab = command.as_table();
        if ((*tab)["type"].value<std::string>().value() == "Read") {
            std::string file = (*tab)["file"].value<std::string>().value();
            command_sequence.push_back(std::make_unique<Read>(file));
        } else if((*tab)["type"].value<std::string>().value() == "DisableServer")
    ↪  {
            int server = (*tab)["server"].value<int>().value();
            command_sequence.push_back(std::make_unique<DisableServer>(server));
        } else if((*tab)["type"].value<std::string>().value() == "EnableServer")
    ↪ {
            int server = (*tab)["server"].value<int>().value();
            command_sequence.push_back(std::make_unique<EnableServer>(server));
        } else {
            std::string file = (*tab)["file"].value<std::string>().value();
            std::string content = (*tab)["content"].value<std::string>().value();
            command_sequence.push_back(std::make_unique<Write>(file, content));
        }
    }
}

const std::map<std::string, std::string> &Client::JobSequence::getInitialFiles()
    ↪ const {
    return initial_files;
}

const std::vector<std::unique_ptr<Command>> &Client::JobSequence::
    ↪ getCommandSequence() const {
    return command_sequence;
}
```

Файл Server.h

```
#ifndef DIST_TASK_SERVER_H
#define DIST_TASK_SERVER_H


#include <map>
#include <memory>
#include "Commands.h"

class Server {
public:
    Server() : files{} {}
    void run();
private:
    class File {
    public:
        File() : content() {}

        [[nodiscard]] const std::string &getContent() const;

        [[nodiscard]] int getVersion() const;

        void setContent(const std::string &content_, int version_);

    private:
        std::string content;
        int version = 0;
    };

    static std::unique_ptr<Command> receiveCommand();
    void processCommand(Command *command);

    std::map<std::string, File> files;
    bool disabled = false;
};


#endif //DIST_TASK_SERVER_H
```

Файл Server.cpp

```
#include <iostream>
#include <mpi.h>
#include <vector>
#include "Server.h"
#include "MPIHelper.h"

void Server::run() {
    std::cout << "Server started" << std::endl;
    while (true) {
        auto command = receiveCommand();
        if (command->getType() == CommandType::CommandStopServer)
            break;
        processCommand(command.get());
    }
}

std::unique_ptr<Command> Server::receiveCommand() {
    MPI_Status status;
    // Probe for message to get buffer size
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);
    int buf_len = 0;
    // Get buffer size, before receiving
    MPI_Get_count(&status, MPI_PACKED, &buf_len);

    std::vector<uint8_t> buf(buf_len);
    MPI_Recv(buf.data(), buf_len, MPI_PACKED, 0, 0, MPI_COMM_WORLD, &status);

    int pos = 0;
    int message_type = MPI_unpack_int(buf, &pos);

    switch ((CommandType)message_type) {
        case CommandRead: {
            auto filename = MPI_unpack_string(buf, &pos);

            return std::make_unique<Read>(filename);
        }
        case CommandWrite: {
            int version = MPI_unpack_int(buf, &pos);

            auto filename = MPI_unpack_string(buf, &pos);

            auto content = MPI_unpack_string(buf, &pos);

            return std::make_unique<Write>(version, filename, content);
        }
        case CommandGetVersion: {
            auto filename = MPI_unpack_string(buf, &pos);

            return std::make_unique<GetVersion>(filename);
        }
        case CommandDisableServer:
            return std::make_unique<DisableServer>(0);
        case CommandEnableServer:
            return std::make_unique<EnableServer>(0);
```

```
        case CommandStopServer:
            return std::make_unique<StopServer>(0);
    }
}

void Server::processCommand(Command *command) {
    switch (command->getType()) {
        case CommandRead: {
            if (disabled) {
                int pos = 0;
                MPIPackBufferFactory bufferFactory{};
                bufferFactory.addInt(1); // Error code

                std::vector<uint8_t> buf = bufferFactory.getBuf();

                int error = -1;
                MPI_pack_int(error, buf, &pos);

                MPI_Send(buf.data(), pos, MPI_PACKED, 0, 0, MPI_COMM_WORLD);
            } else {
                auto commandRead = dynamic_cast<Read *>(command);
                const std::string &text = files[commandRead->getFilename()].
    ↪ getContent();
                int version = files[commandRead->getFilename()].getVersion();

                int pos = 0;

                MPIPackBufferFactory bufferFactory{};
                bufferFactory.addInt(2); // Type of message
                bufferFactory.addString(text);

                std::vector<uint8_t> buf = bufferFactory.getBuf();

                MPI_pack_int(version, buf, &pos);
                MPI_pack_string(text, buf, &pos);

                MPI_Send(buf.data(), pos, MPI_PACKED, 0, 0, MPI_COMM_WORLD);
            }
            break;
        }
        case CommandWrite: {
            int response = 0;
            if (disabled) {
                response = 1;
            } else {
                auto commandWrite = dynamic_cast<Write *>(command);
                files[commandWrite->getFilename()].setContent(commandWrite->
    ↪ getContents(), commandWrite->getVersion());
            }

            MPI_Send(&response, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            break;
        }
        case CommandGetVersion: {
            int version;
            if (disabled) {
```

```
                version = -1;
            } else {
                auto commandGetVersion = dynamic_cast<GetVersion *>(command);
                version = files[commandGetVersion->getFilename()].getVersion();
            }
            MPI_Send(&version, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            break;
        }
        case CommandDisableServer: {
            disabled = true;
            break;
        }
        case CommandEnableServer: {
            disabled = false;
            break;
        }
        case CommandStopServer:
            break;
    }
}

const std::string &Server::File::getContent() const {
    return content;
}

int Server::File::getVersion() const {
    return version;
}

void Server::File::setContent(const std::string &content_, int version_) {
    content = content_;
    version = version_;
}
```

Исходный код для отказоустойчивой MPI программы

Файл main.cpp

```
#include <iostream>
#include <mpi.h>
#include <mpi-ext.h>
#include <cstdlib>
#include <vector>
#include <csignal>

#include "types.h"
#include "functions/arcsin.h"
#include "functions/heaviside_step.h"
#include "functions/exp.h"
#include "messaging.h"

MPI_Comm main_comm = MPI_COMM_WORLD;
int num_procs, mpi_proc_id; // Current number of processes and id of current
    ↪ process
int num_procs_initial; // Number of processes before any failures
std::vector<int> failures{};

void verbose_errhandler(MPI_Comm *pcomm, int *perr, ...) {
    int err = *perr;
    char errstr[MPI_MAX_ERROR_STRING];
    int i, size, nf=0, len, eclass;
    MPI_Group group_c, group_f;
    int *ranks_gc, *ranks_gf;

    MPI_Error_class(err, &eclass);
    if( MPIX_ERR_PROC_FAILED != eclass ) {
        MPI_Abort(*pcomm, err);
    }

    MPI_Comm_rank(*pcomm, &mpi_proc_id);
    MPI_Comm_size(*pcomm, &size);

    MPIX_Comm_failure_ack(*pcomm);
    MPIX_Comm_failure_get_acked(*pcomm, &group_f);
    MPI_Group_size(group_f, &nf);
    MPI_Error_string(err, errstr, &len);

    printf("Rank %d / %d: Notified of error %s. %d found dead: { ", mpi_proc_id,
    ↪ size, errstr, nf);

    MPI_Comm_group(*pcomm, &group_c);
    ranks_gf = (int*)malloc(nf * sizeof(int));
    ranks_gc = (int*)malloc(nf * sizeof(int));
    for(i = 0; i < nf; i++)
        ranks_gf[i] = i;

    MPI_Group_translate_ranks(group_f, nf, ranks_gf, group_c, ranks_gc);
```

```
    for (i = 0; i < nf; i++) {
        printf("%d ", ranks_gc[i]);
        if (ranks_gc[i] == 0) {
            printf("}\nCritical. Master died. Slaves will terminate\n");
            MPI_Abort(*pcomm, err);
        }
        failures.push_back(ranks_gc[i]);
    }
    printf("}\n");
    free(ranks_gf); free(ranks_gc);
    MPIX_Comm_shrink(*pcomm, &main_comm);
    MPI_Comm_rank(main_comm, &mpi_proc_id);
    MPI_Comm_size(main_comm, &num_procs);
}

/**
 * Compound function with heavy calculations
 */
data_t func_big(data_t x) {
    return exp(x) + heaviside_step(x) + arcsin(x);
}

bool invalidate = false;

class Result {
public:
    void timer_start() {
        time = MPI_Wtime();
    }

    void timer_end() {
        time = MPI_Wtime() - time;
    }

    void set_result(data_t res) {
        result = res;
    }

    [[nodiscard]] data_t get_result() const {
        return result;
    }

    [[nodiscard]] double get_runtime() const {
        return time;
    }
private:
    double time;
    data_t result;
};

data_t run(Partition *x_, const func_type func) {
    data_t result = 0.0;

    size_t n = x_->get_n();
    for (size_t i = 0; i < n; i++) {
        result += func(x_->get(i));
```

```
    }

    return result;
}

Result *run_full(const size_t n, const data_t a, const data_t b, const func_type
    ↪ func) {
    auto *res = new Result();
    res->timer_start();

    auto *full = new Partition(a, b, n);

    size_t partition_step = n / num_procs;
    std::vector<Partition> partitions{};
    for (size_t i = 0, j = 0; j < num_procs; j++) {
        if (j < n % num_procs) {
            partitions.emplace_back(full->get(i), full->get(i + partition_step +
    ↪ 1), partition_step + 1);
            //printf("Partition %lu %lu %lu\n", i, i + partition_step + 1,
    ↪ partition_step + 1);
            i += partition_step + 1;
        } else if (partition_step != 0) {
            partitions.emplace_back(full->get(i), full->get(i + partition_step),
    ↪ partition_step);
            //printf("Partition %lu %lu %lu\n", i, i + partition_step,
    ↪ partition_step);
            i += partition_step;
        }
    }
    partitions[0] = Partition(full->get(0), full->get(partition_step + (n %
    ↪ num_procs > 0)),
                      partition_step + (n % num_procs > 0));
    data_t r = 0.0;

    // While there are unfinished partitions
    do {
        int num_running = num_procs;
        for (size_t i = 1; i < num_running; i++) {
            // Send all but first to slaves
            if (i < partitions.size())
                master_send_job(&partitions[i], (int) i);
            else
                // All other slaves are ordered to idle (return 0.0 to reduce)
                master_send_idle((int) i);
        }
        // Process first job
        r += run(&partitions[0], func);

        // Get all results from live processes
        r = reduce(r);

        // Remove all successfully calculated partitions
        std::vector<Partition> nextPartitions{};
        printf("Unfinished jobs {");
        for (size_t i = 0; i < partitions.size(); i++) {
```

```
            if (std::find(failures.begin(), failures.end(), i) != failures.end()
    ↪ || i >= num_running) {
                printf("%lu, ", i);
                nextPartitions.push_back(partitions[i]);
            }
        }
        printf("}\n");
        partitions = nextPartitions;

        failures.clear();
    } while (!partitions.empty());

    printf("All done!\n");

    r += (func(full->get(n)) - func(full->get(0))) / 2;
    r *= full->get_delta();

    res->set_result(r);
    res->timer_end();

    return res;
}

void actual(const size_t n, const data_t a, const data_t b, const func_type func)
    ↪  {
    Result *res = run_full(n, a, b, func);

    std::cout << "Runtime: " << res->get_runtime() << " seconds. ";
    std::cout << "Result: " << res->get_result() << std::endl;

    delete res;
}

void run_slave(int argc, char *argv[]) {
    size_t kill_proc_count = 0;
    if (argc == 3) {
        char *end;
        size_t cnt = strtoull(argv[2], &end, 10);
        if (end != argv[2])
            kill_proc_count = cnt;
    }

    while (true) {
        Partition *p;
        int err = slave_receive_job(&p);
        if (err == 2)
            return;

        data_t res = 0.0;

        // Kill last kill_proc_count processes. Killing last is important as to
    ↪ kill them only once
        if (mpi_proc_id >= num_procs_initial - kill_proc_count)
            raise(SIGKILL);

        if (err == 0) {
```

```
            const func_type func = func_big;
            res = run(p, func);
            delete p;
        }

        reduce(res);

        // Invalidate coefficient table
        invalidate = true;
        arcsin(0);
        exp(0);
        heaviside_step(0);
        invalidate = false;
    }
}

void run_master(int argc, char *argv[]) {
    size_t n = 100000;
    if (argc == 2) {
        char *end;
        size_t cnt = strtoull(argv[1], &end, 10);
        if (end != argv[1])
            n = cnt;
    }
    const data_t a = -1.0;
    const data_t b = 1.0;
    const func_type func = func_big;

    actual(n, a, b, func);

    int num_process;
    MPI_Comm_size(main_comm, &num_process);
    for (int i = 1; i < num_process; ++i) {
        master_send_terminate(i);
    }
}

int main(int argc, char *argv[]) {
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(main_comm, &mpi_proc_id);
    MPI_Comm_size(main_comm, &num_procs);
    num_procs_initial = num_procs;

    MPI_Errhandler errh;
    MPI_Comm_create_errhandler(verbose_errhandler, &errh);
    MPI_Comm_set_errhandler(main_comm, errh);

    if (mpi_proc_id != 0)
        run_slave(argc, argv);
    else
        run_master(argc, argv);

    MPI_Barrier(main_comm);
    MPI_Finalize();
}
```