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

Отчет по учебному курсу «Распределенные системы»

Автор: Губанов Федор

группа 420

Москва, 2021 г.

Содержание

Постановка задачи

Требуется сделать следующее:

- Реализовать программу, моделирующую выполнение протокола голосования для 11 файловых серверов при помощи пересылок MPI типа точка-точка.

- Доработать MPI-программу, реализованную в рамках курса “Суперкомпьютеры и параллельная обработка данных”. Добавить контрольные точки для продолжения работы программы в случае сбоя. Реализовать один из 3-х сценариев работы после сбоя:
    1. продолжить работу программы только на “исправных” процессах;
    2. вместо процессов, вышедших из строя, создать новые MPI-процессы, которые необходимо использовать для продолжения расчетов;
    3. при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

После реализации программы, моделирующую выполнение протокола голосования, необходимо оценить сколько времени потребуется для выполнения одним процессом 3-х операций записи и 10 операций чтения N байтов информации с файлом, расположенным (размноженным) на 11 серверах. А также определить оптимальные значения кворума чтения и кворума записи для N=300. Время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1).

Реализация программы и оценка времени выполнения

Идея метода голосования - запрашивать чтение и запись файла у многих серверов.
Для реализации этого метода используется кворум чтения - далее N_r и кворум записи - далее N_w.

1. Для выполнения чтения достаточно обратиться к N_r серверам и использовать данные того сервера, который располагает последней версией файла.
2. Для выполнения записи необходимо, чтобы все N_w сервером выполнили запись и увеличили свои версии файлов на 1. При этом у всех серверов должно быть согласие относительно номера текущей версии.

Для обеспечения корректности данных значения для кворума должны удовлетворять неравенству: N_r + N_w > N, где N - количество всех серверов. Поскольку чтение является более частотной операцией, то естественно взять N_r = 1.
Данный алгоритм был реализован с помощью функций MPI_Send и MPI_Recv. А также работа с директориями/файлами с помощью стандартных библиотек C++.
Репозиторий с кодом и файлами сборки/запуска:
https://github.com/Fedor1533/MSU_Dis_Systems/tree/main/Task1

Оценка времени выполнения

Оценим время работы алгоритма. Если время старта равно 100, время передачи байта равно 1 (Ts=100,Tb=1), то время выполнения операции передачи N байт рассчитывается следующим образом:

$$time = Ts + N \cdot Tb$$

Тогда для десяти операций чтения потребуется отправить N_r серверам запрос на чтение, после чего каждый из N_r серверов пришлет версию файла, которую он хранит. Предположим каждая из посылок содержит 4 байта. В итоге время выполнения 10 операций чтения:

time1 = 10(N_r*Ts + N_r*4*Tb)* - запрос на чтение
time2 = 10(N_r*Ts + N_r*4*Tb)* - посылка локальной версии с сервера
*timeR = time1 + time2 = 2000*N_r + 80*N_r*

Для трех операций записи *N* байт потребуется отправить *N_w* серверам запрос на запись, отправить *N* байт *N_w* серверам, после чего каждый из *N_w* серверов отправляет новую версию, подтверждая изменения, при этом версии должны быть согласованы.
В итоге время выполнения 3 операций записи:

time1 = 3(N_w*Ts + N_w*4*Tb)* - запрос на запись
time2 = 3(N_w*Ts + N_w*N*Tb)* - отправка *N* байт серверам
time3 = 3(N_w*Ts + N_w*4*Tb)* - посылка локальной версии с сервера подтверждающая запись
*timeW = time1 + time2 + time3 = 900*N_w + (3*N+24)*N_w*

Оптимальное значение

Определим оптимальные значения кворума чтения и кворума записи для N=300. Для этого используем тождества, полученные при вычислении времени выполнения операций чтения и записи.
В результате, задача сводится к минимизации данного выражения:

*2000*N_r + 80*N_r + 900*N_w + 924*N_w , при этом N_w+N_r > 11*

Минимум достигается при *N_w = 11 и N_r = 1,* то есть согласие на запись должны дать все серверы, а на чтение один любой процесс.

Доработка MPI-программы для возможности продолжения работы в случае сбоя

Для реализации возможности продолжения работы программы без ошибок в случае сбоя одного из процессов, необходимо было написать собственный обработчик ошибок, который будет срабатывать в таких ситуациях.
В стандарте MPI существует для этого специальные функции *MPI_Comm_create_errhandler* и *MPI_Comm_set_errhandler*. Однако этот функционал в стандартной библиотеке ограничен и не позволяет определить процесс, в котором произошла ошибка. В результате было использовано расширение MPI – ULFM.

В программе были сделаны определенные доработки:

1. С помощью фунцкций *MPI_Comm_create_errhandler* и *MPI_Comm_set_errhandler* добавлен обработчик ошибок err_handler.
2. В качестве основного решения был выбран вариант, когда при запуске программы создается резервный процессов, который сможет заменить убитый процесс. В данной реализации - последний процесс считается резервным.
3. Данная реализация не предполагает, что резервный процессы даст сбой.
4. Для всех процессов сформировано имя файла для записи данных контрольных точек.
5. В начале данные распределяются по процессам с помощью операций *MPI_Isend* и *MPI_Recv*, при этом последнему процессу данные не отправляются.
6. Далее происходит вызов функции `kernel_fdtd_2d`, в которой перед каждой итерацией алгоритма процессы читают данные из файла, работают с ними и в конце итерации записывают актуальную версию в файл, при этом в течении итерации происходит обмен данными между процессами. При возникновении ошибки все процессы заново считывают данные из файла и начинают итерацию сначала.

7. Для того чтобы процессы находились на итерациях с одинаковым номером, были расставлены «барьеры» с помощью функции *MPI_Barrier*.

8. На определенной итерации один из процессов умирает, после чего во всех процессах управление переходит в функцию обработчик ошибок.

9. В обработчике ошибок на базе старого коммуникатора создается новый, не включающий в себя вышедший из строя процесс. Это реализуется с помощью функции, не входящей в стандарт MPI - *MPIX_Comm_shrink*.

10. После создания нового коммуникатора в обработчике ошибок, все работающие процессы получает новый ранг, при этом далее они возможно будут работать с другим файлом, данное изменение не влияет на результат выполнения алгоритма.

11. Работа программы продолжается на оставшихся процессах и результат собирается в нулевом процессе с помощью операций *MPI_Isend* и *MPI_Irecv*.

Для запуска программы рекомендуется использовать Docker для использования библиотеки ULFM. Код решения и все необходимые для запуска файлы находятся в репозитории по ссылке:

https://github.com/Fedor1533/MSU_Dis_Systems/tree/main/Task2

Заключение

Таким образом, была реализована программа, моделирующую выполнение протокола голосования для 11 файловых серверов при помощи пересылок MPI типа точка-точка.
Оценено время выполнения одним процессов 3-х операций записи и 10 операций чтения N байтов информации с файлом, расположенным (размноженным) на 11 серверах. Определено оптимальное значение кворума чтения и кворума записи для N=300.
Также была доработана MPI-программа, реализованная в рамках курса “Суперкомпьютеры и параллельная обработка данных”, которая стала устойчива к сбоям в процессах.