Экспериментальное качественное сравнение двух подходов к динамической балансировке нагрузки

- Мустафин Дамир Эркинович, ФИТ НГУ, 2 курс
- Мичуров Михаил Антонович, ФИТ НГУ, 2 курс

Руководитель: Власенко Александр Юрьевич

Цель

Реализовать и на наборе тестов сравнить следующие подходы к динамической балансировке нагрузки:

- **централизованная стратегия балансировки** (наличие выделенного «процесса-диспетчера», перераспределяющего нагрузку между «процессами-работниками»);

- **распределенная стратегия балансировки** (каждый освободившийся процесс обращается за очередной порцией работы к соседу).

Формулировка задачи

Дано:

• Множество задач (f[1]…f[N]) со случайными весами в определенном диапазоне.

• Каждая из задач генерирует свой фрагмент данных (f[i] -> a[i]). Все фрагменты данных имеют одинаковый размер в байтах.

• Множество процессов p[1], …, p[K]. Процессы выполняют задачи f[1]…f[N], разделяя работу между собой.

• Каждая из задач f[i] может быть либо независимой, и тогда ее можно вычислять сразу, либо зависимой от произвольного количества фрагментов данных $a[j_1]$, $a[j_2]$, …, $a[j_k]$, т.ч. $\max(j_1, \ldots, j_k) < i$.

Формулировка задачи

Необходимо получить весь результирующий набор фрагментов данных a[1],…,a[N], динамически перераспределяя вычисление задач f[1], …, f[N] по процессам p[1], …, p[K].

Написать 2 программы:

- 1-я - на основе централизованной балансировки (процесс p[K] занимается динамической балансировкой нагрузки между процессами p[1]…p[K-1]);
- 2-я - на основе распределенной балансировки (все процессы p[0]…p[K] выполняют работу, при этом нагрузка передается освобождающимся процессам от соседних).

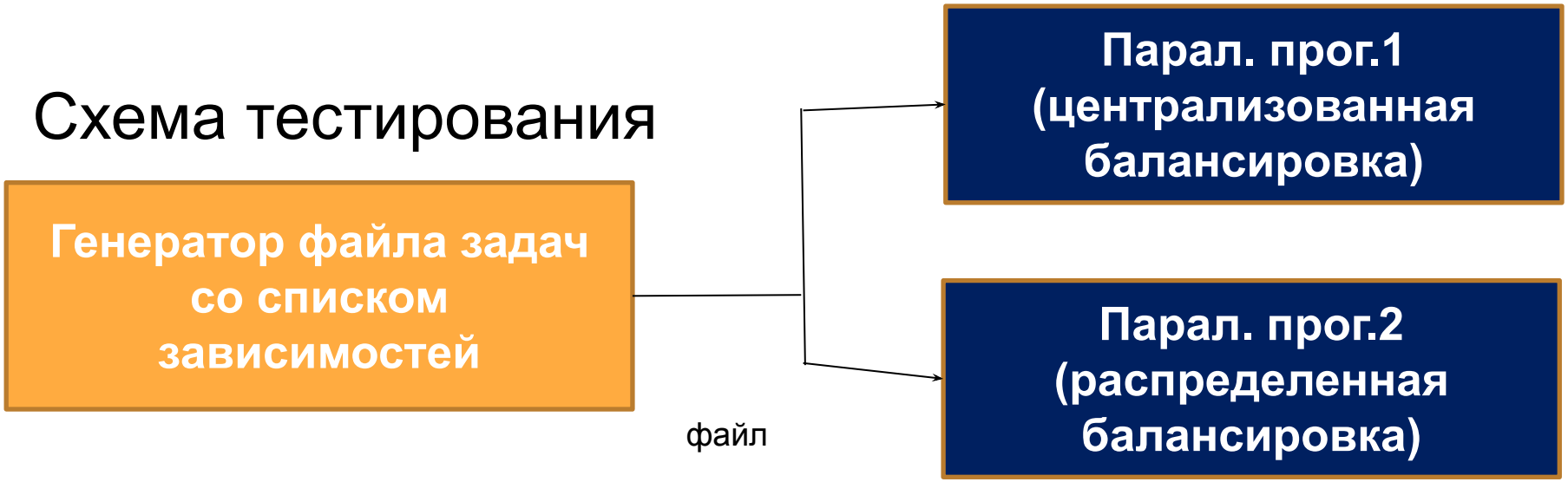
Сравнить эффективность двух программ на наборе тестов, варьируя параметры:

- количество задач N;
- диапазон весов задач;
- объем генерируемых фрагментов данных a[i];
- количество процессов K.

Требования

- Каждый процесс-работник двухпоточный. Один поток занимается вычислениями задач f[i], другой – управляющими действиями: запрос дополнительной работы от других и, наоборот, отправка части своей работы другим.
- Использовать упреждающую подгрузку работы (до окончания выполнения всех своих задач процесс-работник запрашивает дополнительную работу у других).

Схема тестирования



Генератор принимает параметры:

- кол-во задач;
- доля независимых задач;
- макс. число зависимостей;
- диапазон весов задач.

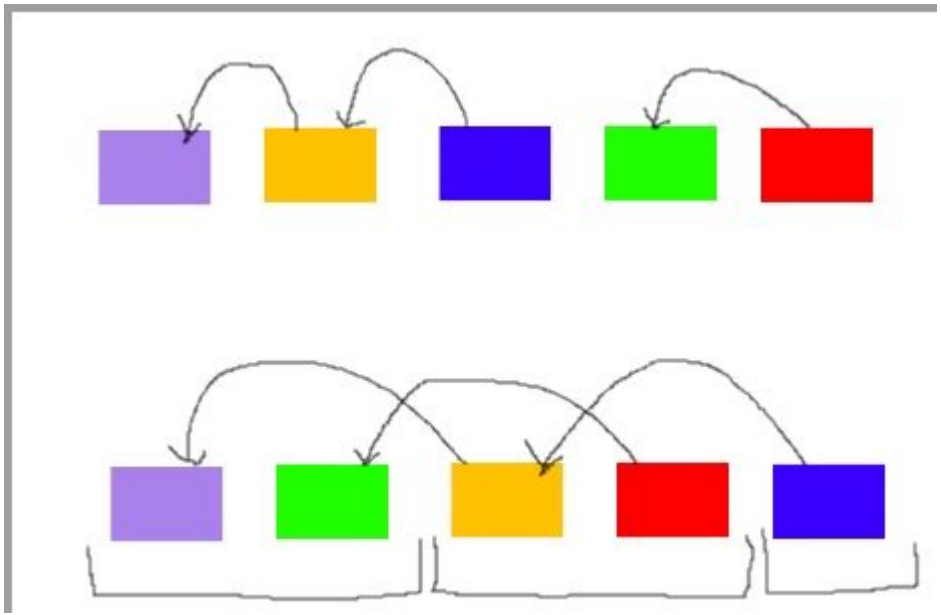
Программы принимают:

- кол-во процессов
- объем фрагмента данных
- **par1** - число оставшихся задач у процесса, по достижении которого нужно запрашивать еще
- **par2** - доля от числа оставшихся задач, которую нужно отправить в ответ на запрос задач

Идея решения (слайд 1 из 5)

- Задачи разбиваются на упорядоченные группы по следующему принципу
  - Задачи, не имеющие зависимостей, попадают в группу 0
  - Задачи внутри группы i + 1 не имеют зависимостей от задач из этой же группы, но имеют как минимум одну зависимость от задачи группы i
- Таким образом, задачи из одной группы можно выполнять параллельно, но перед выполнением задач из группы i + 1 должны быть выполнены задачи из групп 0 … i
- Список задач выполняется итерационно (1 итерация – 1 группа)

Пример разбиения на группы



Идея решения (слайд 2 из 5)

Централизованный алгоритм

1. Выделенный процесс считывает граф зависимостей из файла, разбивает задачи на группы и каждую группу поровну разделяет между всеми процессами
2. На каждой итерации

    Служебный поток процесса-работника

    i. Проверяет зависимости задач, которые ему предстоит выполнить на текущей итерации, и запрашивает у диспетчера недостающие номера фрагментов данных.
    ii. После получения номеров фрагментов данных и номеров процессов, у которых они есть, начинает их запрашивать.
    iii. Отдает/получает запрошенное число задач и необходимые фрагменты данных для их подсчета
    iv. Уведомляет диспетчера о выполнении задачи

Идея решения (слайд 3 из 5)

Централизованный алгоритм

Диспетчер

i. При получении от процесса информации о номере выполненной задачи, записывает к себе в таблицу. Таким образом, диспетчер знает, какие фрагменты данных имеются и сколько задач осталось у каждого из процессов
ii. Сообщает процессам о начале итерации/завершении работы
iii. Если у процесса осталось менее **par1** задач, находит наиболее нагруженный процесс, и отправляет процессу информацию о том, сколько задач следует забрать у нагруженного процесса ((число оставшихся задач) * **par2** задач)
iv. При получении запроса на фрагменты данных, отправляет информацию о том, у каких процессов их можно взять

Идея решения (слайд 4 из 5)

Распределенный алгоритм

1. Выделенный процесс считывает граф зависимостей из файла, разбивает задачи на группы и каждую группу поровну разделяет между всеми процессами
2. На каждой итерации

    Основной поток

    i. Основной поток проверяет зависимости задач, которые ему предстоит выполнить на текущей итерации, и запрашивает у других процессов недостающие фрагменты данных
    ii. Пока есть задачи, основной поток их выполняет (если зависимости еще не пришли - ждет); если задач осталось менее **par1**, он запрашивает дополнительные задачи у следующего процесса, и не ждет ответа; если же задач не осталось, то после отправки запроса основной поток ждет ответа (дополнительные задачи, либо уведомление о том, что задач больше нет)
    iii. После выполнения очередной задачи основной поток уведомляет об этом другие процессы. Таким образом, процессы знают, где можно взять интересующие их фрагменты данных

Идея решения (слайд 5 из 5)

Распределенный алгоритм

Служебный поток

i. Служебный поток принимает запросы и уведомления от других процессов и от основного потока
ii. В ответ на запрос на дополнительные задания он отправляет (число оставшихся задач) * **par2** задач
iii. При приеме дополнительных задач запрашивает отсутствующие зависимости
iv. При получении уведомления о выполнении задания сохраняет информацию о том, где можно взять вычисленный фрагмент данных
v. Также служебный поток принимает и отправляет фрагменты данных

Выходные параметры программ

- Общее время работы, с
- Макс. время выполнения одной итерации, с
- Мин. время выполнения одной итерации, с
- Макс. процент относительного дисбаланса за итерацию ($d_{i\ отн.\ max}$)
- Мин. процент относительного дисбаланса за итерацию ($d_{i\ отн.\ min}$)
- Макс. доля ожидания и коммуникаций
- Мин. доля ожидания и коммуникаций

Примечание

Относит. дисбаланс: $d_{i\ отн.} = \frac{d_i * t_{i\ max}}{t_{1\ max} + t_{2\ max} + \cdots + t_{n\ max}}$

Абсолют. дисбаланс: $d_i = (t_{i\ max} - t_{j\ min}) / t_{i\ max}$

Реализация

При реализации использованы следующие инструменты:

1. c99 в качестве единственного ЯП
2. POSIX Threads для создания потоков и обеспечения взаимодействия между ними
3. MPI для межпроцессного взаимодействия

Получилось реализовать:

- Процессы-работники двухпоточные
- Дополнительные задачи подгружаются заранее (регулируется параметром)

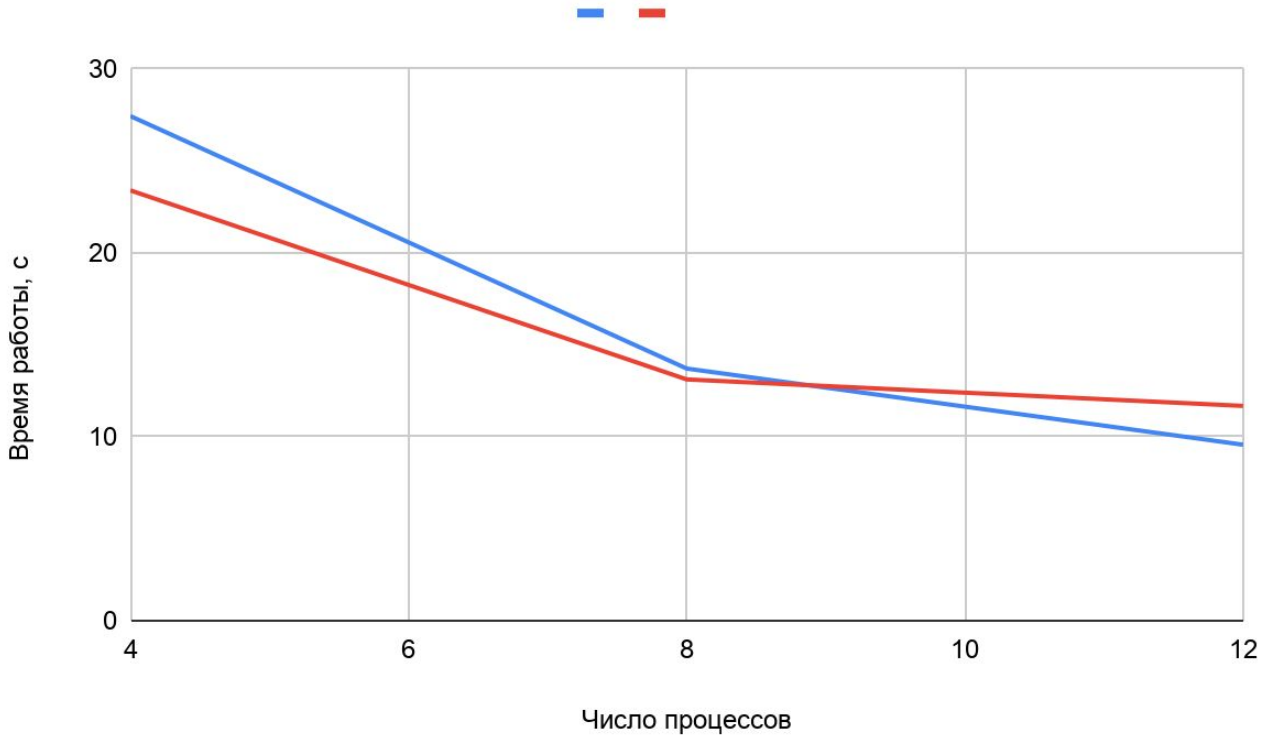
Не получилось:

- Не удалось полностью разделить выполнение основной работы и служебных задач: основной поток, помимо вычислений, оповещает процессы о выполнении задач, запрашивает данные и дополнительные задачи

Тестирование

Число задач: 480
Доля независимых: 0.25
Максимальное числ зависимостей: 5
Диапазон веса задачи: [50; 200)

Синяя линия - централизованная балансировка, красная - распределенная



Заключение

Централизованная балансировка показывает себя лучше на большом числе процессов, распределенная - на малом. На большом числе процессов распределенная балансировка не только не уменьшает дисбаланс, но даже увеличивает его, хотя общее время работы уменьшается.

Благодарим за внимание!

Дополнительные слайды

Структура входного файла, описывающего граф зависимостей

Первая строка: число задач N

Следующие N строк:
вычислительный вес задачи и
номера задач, от которых зависит
данная, разделенные пробелами

```
5
13
17
10 1 2
19 1
20 2 3 4
```

Пример входного файла

Ссылки

Описание алгоритма централизованной балансировки:

https://docs.google.com/document/d/1opBow_9oSTnFeCe6MVtat7srr6YGEHwGa8Lc-iL5ki8/edit

Блок-схема алгоритма распределенной балансировки:

https://drive.google.com/file/d/119DsSnV79bLkCpbPwY_36Av4V7XICNdE/view