Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Горошко Артём Дмитриевич

Балансировка сегментов потока в многоядерной системе обработки данных

Курсовая работа

Научный руководитель:
Ассистент, к.ф.-м.н.
Степанов Евгений Павлович

Москва, 2024

Аннотация

В работе рассматривается задача балансировки сегментов потока в многоядерной системе обработки пакетов. Актуальность задачи обусловлена необходимостью равномерно распределять поступающую нагрузку по отдельным ядрам, в то же время сохраняя исходный порядок поступления пакетов внутри транспортного потока. Проведен обзор методов балансировки, удовлетворяющих поставленной задаче. В работе предложены и реализованы алгоритмы балансировки сегментов потока, а также проведено сравнение средней загруженности обработчиков.

Содержание

1 Введение

1.1 Актуальность задачи

Объем трафика в сети с каждым днем растет: согласно данным МСЭ - до сих пор мировой интернет-трафик рос в среднем на 22 % в год, увеличившись с 2401 эксабайта в 2019 году до 5291 эксабайта в 2022 году [1]. С увеличением трафика увеличивается и нагрузка на системы обработки пакетов. Балансировка нагрузки важна для обеспечения эффективного использования вычислительных ресурсов в многоядерных системах:

1. Балансировка нагрузки позволяет минимизировать простои ядер.
2. С помощью балансировки нагрузки достигается более высокая суммарная производительность обработчиков (обрабатывается больше пакетов за одинаковое время).
3. Балансировка нагрузки позволяет уменьшить задержку обработки пакетов.

При балансировке нагрузки на основе пакетов может возникать переупорядочивание. Оно появляется при возникновении очередей на обработчиках - к примеру первый пакет какого-то потока отправляется на один обработчик, очередь которого практически забита, а второй распределен на чуть менее загруженный обработчик, в связи с чем быстрее пройдет этапы обработки и опередит первый пакет, нарушив порядок поступления. Если второй пакет опережает первый, приложение посчитает пакет потерянным и отправит повтоный запрос пакета и как итог - скорость поступления пакетов в пространство приложений падает из-за работы алгоритмов управления перегрузкой [2]. Балансировка нагрузки на основе потоков решает проблему переупорядочивания, но часто приводит к неоптимальной производительности. Когда надо распределить продолжительные потоки, которые будут поступать на одно и то же ядро, это может привести к перегрузке, если их интенсивность придется на одно и то же время. Балансировка нагрузки с использованием сегментов потока позволяет избегать этих проблем.

Сегмент потока (от англ. flowlet) [3] - это последовательность таких пакетов транспортного потока, что интервал времени между соседними пакетами меньше заранее заданной величины δ (см. рис. 1).

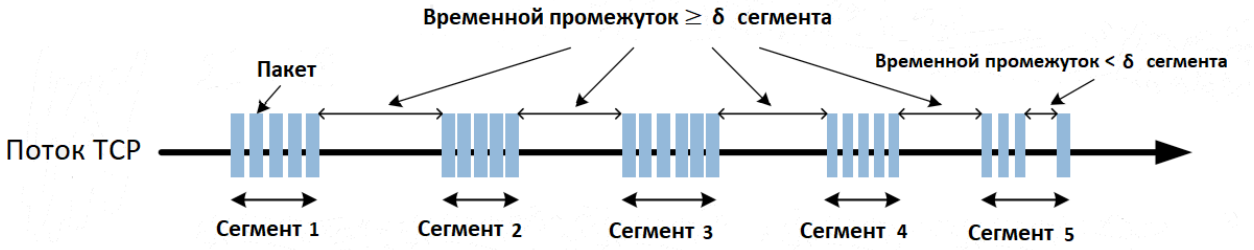

Рис. 1: Сегменты потока [3]

Из определения сегмента потока несложно догадаться, что проблема с переупорядочиванием при использовании балансировки сегментов не возникает. Деление же потоков на сегменты позволяет добиваться равной загруженности обработчиков, что помогает справляться с неоптимальной производительностью.

Стоит добавить, что наибольший выигрыш при такой балансировке достигается при большом количестве одновременно поступающих потоков - в случае одного потока, разделяемого на сегменты, на каждом этапе обработки одновременно будет работать максимум по одному обработчику на сегмент потока. При параллельном поступлении нескольких потоков в систему будут поступать одновременно уже несколько сегментов, а не по одному, в результате чего меньше ядер будут простаивать.

В качестве многоядерной системы обработки пакетов в работе рассматривается DPDK (Data Plane Development Kit) [4] - специализированный набор библиотек и драйверов, позволяющий быстро обрабатывать пакеты в пользовательском пространстве. Быстрая обработка достигается за счет обхода сетевого стека TCP/IP в ядре Linux.

Обработка пакетов в Linux происходит следующим образом [5]: когда пакет поступает на сетевую карту, он копируется оттуда в основную память с помощью механизма DMA — Direct Memory Access. После этого требуется сообщить системе о появлении нового пакета и передать данные дальше, в специально выделенный буфер (эта структура выделяется для каждого пакета и освобождается, когда пакет попадает в пользовательское пространство). Для этой цели в Linux используется механизм прерываний: прерывание генерируется всякий раз, когда новый пакет поступает в систему. Затем пакет ещё нужно передать в пользовательское пространство.

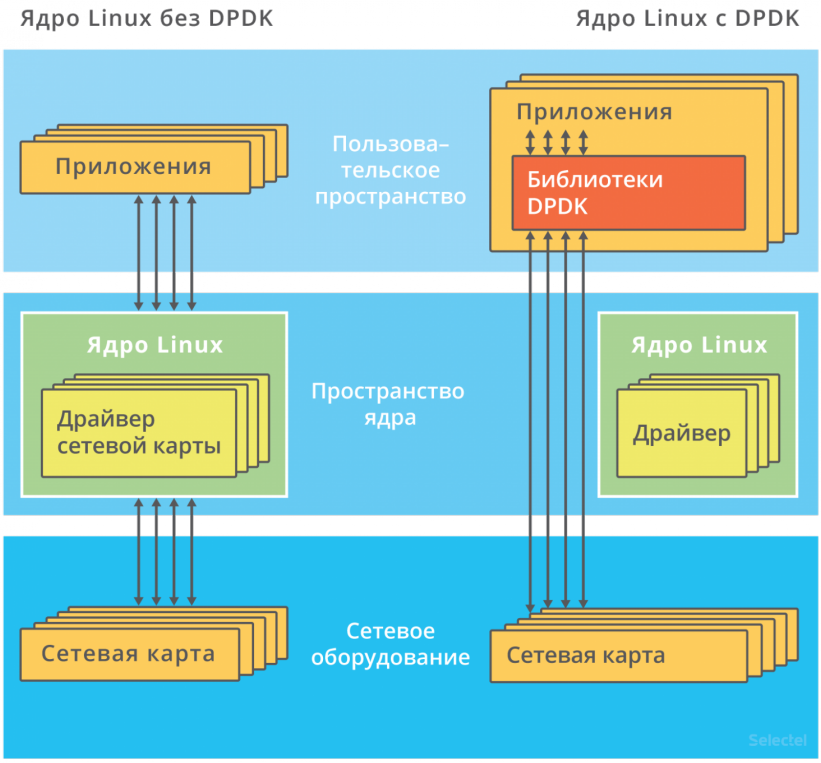

Рис. 2: Процессы обработки пакетов в Linux и в DPDK [6]

В случае использования DPDK ядро Linux не задействовано вообще: взаимодействие с сетевой картой осуществляется через специализированные драйверы и библиотеки. Основные стадии обработки пакетов с использованием DPDK приведены ниже:

1. Поступившие пакеты попадают в кольцевой буфер (таблица указателей на хранимые в памяти объекты), приложение периодически проверяет этот буфер на наличие новых пакетов.

2. Если в буфере имеются новые дескрипторы пакетов, приложение обращается к буферам пакетов DPDK через указатели в дескрипторах пакетов. Вместо одной большой структуры sk_buff — в DPDK используется много буферов rte_mbuf небольшого размера. Буферы создаются до запуска приложения, использующего DPDK, и хранятся в специально выделенном блоке памяти.

3. Если в кольцевом буфере нет никаких пакетов, то приложение опрашивает на-

ходящиеся под управлением DPDK сетевые устройства (сетевые карты), а затем снова обращается к кольцу (кольцевому буферу).

DPDK обладает следующими преимуществами:

- Высокая производительность. DPDK оптимизирован для обработки пакетов и использует преимущества современных многоядерных процессоров.
- Низкая задержка. DPDK может значительно сократить задержку обработки пакетов, что особенно важно для веб-приложений, требующих отклика в реальном времени.
- Лёгкий дизайн. DPDK прост, избегает сложных абстракций и обработки протоколов в традиционном стеке ядра.
- Масштабируемость. DPDK позволяет пользователям гибко настраивать очереди для обработки пакетов и управлять потоками пакетов.

1.2 Цель и задачи работы

Цель: Разработать метод балансировки сегментов потока по ядрам двухэтапного конвейера обработки пакетов DPDK, обеспечивающий их равномерную загрузку.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Сделать математическую постановку задачи балансировки сегментов потока по ядрам двухэтапного конвейера обработки пакетов DPDK.
2. Провести обзор методов балансировки пакетов, потоков и сегментов потока с целью выбора методов, подходящих к поставленной задаче.
3. Разработать и реализовать алгоритмы балансировки сегментов потока на основе выбранных методов.
4. Исследовать эффективность распределения сегментов потока по ядрам для реализованных алгоритмов балансировки.

1.3 Структура работы

В главе 2 описывается математическая постановка задачи. В главе 3 проводится обзор существующих методов балансировки. В главе 4 описывается стенд, в которому будут реализованы алгоритмы балансировки, разработанные на основе выбранных методов балансировки. В главе 5 представлены результаты экспериментального исследования по сравнению эффективностей методов балансировки сегментов потока между собой и с методами балансировки потоков. Глава 7 подводит заключение настоящей работы.

2 Математическая постановка

Введем основные понятия и термины:

1. Рассматривается временной интервал $T = [0, t_{end}]$, где $t_{end} \in \mathbb{R}^+$;
   Разобьём этот временной интервал на $q \geqslant 1$ различных временных интервалов: $[t_h, t_{h+1}] \subseteq T, h = \overline{1, q}$. Первый интервал: $[t_1, t_2]$, второй: $[t_2, t_3]$ и тд.

2. M_1, M_2 - множества обработчиков первого и второго этапов соответственно, $|M_1| = k_1, |M_2| = k_2$; $M = M_1 \cup M_2$ - множество всех обработчиков, $|M| = k = k_1 + k_2$;

3. P - множество пакетов, поступивших в систему;
   Fl - множество сегментов потоков.

4. $g : P \to Fl$ - функция определения сегмента потока.

5. $\phi : T \times M \to \mathbb{N}$ - функция определения количества пакетов, прошедших через обработчик за определенное время.

6. $\psi_1 : P \times M_1 \to \mathbb{R}^+$ - функция определения времени обработки пакета на первом этапе обработки.
   $\psi_2 : P \times M_2 \to \mathbb{R}^+$ - функция определения времени обработки пакета на втором этапе обработки.
   Обе функции обладают следующим свойством:
   - $\forall p \in P : \psi_\gamma(p, m_i) = \psi_\gamma(p, m_j)$, где $m_i, m_j \in M_\gamma, i \neq j, i, j \in [1, k_\gamma], \gamma \in \overline{1, 2}$
   Другими словами, время обработки любого пакета равное на всех обработчиках.

Таким образом, входными данными к задаче являются:

1. Временной интервал T и его разбиение на q частей.

2. Множества обработчиков M_1 и M_2.

3. Множество пакетов P[1].

4. Функция определения сегмента потока g.

[1]Замечание: пакеты, которые должны поступить в систему, не известны балансировщику заранее

5. Функция определения количества пакетов ϕ.

6. Функции определения времени обработки ψ_1 и ψ_2.

Также введем следующие обозначения и индексы:

- $h \in [1, q]$ - номер временного интервала.
- $\gamma \in [1, 2]$ - номер этапа обработки.
- $i_\gamma \in [1, k_\gamma]$ - номер обработчика с этапа γ.
- $m_{i_\gamma} \in M_\gamma$ - обработчик с номером i_γ.
- $n_{\gamma i_\gamma h} \in [1, \phi_\gamma(t_{h+1} - t_h, m_{i_\gamma})]$ - номер пакета, обработка которого происходила на обработчике m_{i_γ}, началась и закончилась во временном интервале h.

Необходимо ввести еще одну функцию:

$f : P \to M_1 \times M_2$ - функцию балансировщика пакетов по обработчикам, обладающая следующим свойством:
- $\forall \alpha, \beta \in P : g(\alpha) = g(\beta) \to f(\alpha) = f(\beta)$

Задача заключается в том, чтобы найти такую функцию балансировки, которая получит наилучшую эффективность. Под эффективностью понимается уменьшение максимального отклонения загруженности каждого обработчика от их средней загруженности (загруженность при этом считаются за все q временных интервалов). То есть:

$$\min_f(\max_{\gamma, i_\gamma, h}(|\xi_{i_\gamma h} - \overline{\xi_\gamma}|)), \text{ где}$$

- $\xi_{i_\gamma h}$ - загруженность обработчика i_γ в промежуток времени $[t_h, t_{h+1}]$, которая определяется формулой (суммирование ведется по всем пакетам $n_{\gamma i_\gamma h}$, переданным через обработчик i_γ в интервале h):

$$\xi_{i_\gamma h} = \frac{\sum_{n_{\gamma i_\gamma h}}(\psi_\gamma(p_{n_{\gamma i_\gamma h}}, m_{i_\gamma}))}{t_{h+1} - t_h}$$

- $\overline{\xi_\gamma}$ - среднее значение загруженности по всем обработчикам этапа γ и временным интервалам, определяющееся формулой:

$$\overline{\xi_\gamma} = \frac{1}{k_\gamma} \sum_{i_\gamma} \overline{\xi_{i_\gamma}}$$

- $\overline{\xi_{i_\gamma}}$ - среднее значение загруженности на обработчике i_γ по всем временным интервалам, которое определяется формулой:

$$\overline{\xi_{i_\gamma}} = \frac{\sum_w \xi_{i_\gamma w}}{q}$$

3 Обзор существующих методов балансировки

В качестве обозреваемых методов будем рассматривать существующие методы балансировки нагрузки, пакетов, потоков и сегментов потока, которые возможно адаптировать под рассматриваемую задачу. Методы будем сравнивать по следующим критериям:

1. Учитывание методом равнозначности обработчиков.
2. Память, затрачиваемая методом на балансировку.
3. Время, за которое метод осуществляет балансировку.

Методы балансировки, основанные на применении методов машинного обучения, не будут рассматриваться в обзоре, в связи с долгим временем обучения и необходимостью использования размеченных наборов данных [7].

В итоге проведенного обзора выделены следующие методы балансировки:

1. Round Robin
2. Weighted Round Robin
3. Least Connection
4. Weighted Least Connection
5. Resource Based
6. Hash Scheduling
7. Fixed Weighting

3.1 Round Robin

Циклическая балансировка нагрузки (Round Robin [8 - 10]) - это самый простой и наиболее часто используемый алгоритм балансировки нагрузки. Поступающие сегменты потока распределяются по обработчикам простым чередованием. Например, если у вас есть три обработчика: пакеты первого сегмента отправляются на первый обработчик в списке, пакеты второго сегмента - на второй обработчик, третьего сегмента - на третий обработчик, четвертого - на первый обработчик и так далее.

При реализации этого метода придется запоминать маршруты еще не закончившихся сегментов потоков (сегментов, последний пакет которых прибыл $< \delta$ единиц времени назад), чтобы пакеты одного сегмента шли по одинаковому маршруту. Также придется помнить номер последнего обработчика в каждом этапе, которому был присвоен новый сегмент, чтобы осуществлять чередование. Пусть n - максимальное количество сегментов, одновременно проходящих через систему обработки. Для таблицы маршрутов нам нужно помнить минимум три вещи: ID сегмента потока, обработчик первого этапа и обработчик второго этапа, отсюда вытекает оценка по памяти, затрачиваемой на балансировку, O(3n).

Время, затрачиваемое на балансировку, зависит уже от реализации. Нужно для каждого пакета, попадающего на обработчик, проверять, есть ли маршрут для его сегмента, или нет, для этого придется проходиться по таблице сегментов потоков - O(n). А можно составить хэш-функцию, которая ставит в соответствие ID сегмента его маршрут и работает в среднем за O(1).

3.2 Weighted Round Robin

Взвешенный циклический перебор (Weighted Round Robin [8 - 10]) аналогичен циклическому алгоритму балансировки нагрузки, но добавляет возможность распределять входящие сегменты потока по обработчикам в соответствии с возможностями каждого обработчика. Он наиболее подходит для распределения входящих объектов балансировки по набору обработчиков, которые имеют различные вычислительные способности. Балансировщик присваивает вес каждому обработчику на основе выбранных им критериев, которые указывают на относительную способность каждого обработчика обрабатывать объекты.

Итак, например: если обработчик № 1 в два раза мощнее (быстрее обрабатывает тот же пакет) обработчика № 2 и обработчика № 3, он получает больший вес, а обработчики № 2 и № 3 получают одинаковый, меньший вес. Если имеется 5 последовательных сегментов, первые 2 отправляются на обработчик № 1, 3ий отправляется обработчик № 2, 4ый - на обработчик № 3. Затем пакеты 5го сегмента потока будут отправлены на обработчик № 1 и так далее.

Реализация этого метода потребует те же струткуры, что и метод балансировки Round Robin, но при этом добавит еще одну таблицу весов, где каждому обработчику будет соответствовать его вес. Отсюда оценка по памяти: O(3n + k), где k - количество обработчиков по всем этапам.

Время на балансировку будет таким же, как и в первом методе - необходимо только один раз рассчитать, какое количество сегментов будет присваиваться каждому обработчику по кругу. Отсюда временная оценка O(n) или же O(1).

3.3 Least Connections

Балансировка нагрузки с наименьшим количеством подключений (Least Connections [9 - 11]) - это алгоритм динамической балансировки нагрузки, при котором сегменты потока распределяются на обработчик с наименьшим количеством приписанным к нему сегментов потоков на момент получения пакета нового сегмента.

К примеру возьмем систему из трех обработчиков. Пусть в какой-то момент времени к трем обработчикам приписано в сумме 4, 3 и 5 сегментов потоков соответственно, тогда при поступлении пакета из нового сегмента потока он отправится на обработчик 2.

Метод также использует для балансировки таблицу сегментов потоков и дополнительно должен помнить, сколько сегментов приписано каждому обработчику на момент поступления пакета, а значит затраты по памяти: O(3n + k).

Временная оценка добавит в себя в каждом из случаев перебор по количеству сегментов потоков, присвоенных ккаждому обработчику, и будет равна O(n + k) или O(k).

3.4 Weighted Least Connections

Взвешенная балансировка нагрузки с наименьшим количеством подключений (Weighted Least Connections [9]) основана на предыдущем методе балансировки и учитывает различные характеристики обработчиков. Балансировщик присваивает вес каждому обработчику на основе относительной вычислительной мощности и доступных ресурсов

каждого обработчика, а после принимает решения о балансировке сегментов потоков на основе приписанных сегментов и назначенных весов обработчиков.

Например, есть три обработчика с весами 2, 1 и 3 соответственно и с 2, 2 и 3 приписанными сегментами потоков. Поступает пакет нового сегмента. Тогда этот пакет отправится на тот обработчик с наименьшим количеством сегментов, которому присвоен наибольший вес.

К затратам по памяти, как и в предыдущих случаях с добавлением весов, оценка увеличится таблицей весов и станет равна O(3n + 2k).

Время также увеличится перебором по весам. То есть оценка станет равна O(n + 2k) или O(2k).

3.5 Resource Based

Балансировка нагрузки на основе ресурсов (Resource Based [9, 10]) позволяет обработчику принимать решения на основе показателей состояния, получаемых с обработчиков. Под состоянием имеется ввиду время, потраченное обработчиком на процесс обработки пакетов. Балансировщик регулярно проверяет состояние у каждого обработчика, а затем соответствующим образом устанавливает динамический вес обработчика.

Таким образом, метод балансировки сегментов, по сути, выполняет детальную "проверку нагруженности" обработчика. Этот метод подходит в любой ситуации, когда для принятия решений о балансировке нагрузки требуется информация о проверке нагруженности каждого обработчика.

Например: пусть обработчики 1, 2 и 3 обрабатывали пакеты 15, 12 и 20 миллисекунд соответственно, тогда пакеты нового сегмента потока отправятся на второй обработчик.

Затраты по памяти для реализации этого метода схожи с затратами на реализацию метода Least Connections, только в качестве значений второй таблицы используется не количество приписанных в момент сегментов потока, а суммарная нагрузка на обработчики. То есть оценка по памяти вновь O(3n + k).

Временная оценка также схожа с методом Least Connections - O(n + 2k) или O(2k).

3.6 Hash Scheduling

Алгоритм балансировки нагрузки по хэшу (Hash Scheduling [9 - 11]) использует ID сегмента потока для генерации уникального хэш-ключа, который используется для назначения пакетов сегмента потока определенному обработчику. Этот метод балансировки является самым быстрым методом балансировки и не требует больших затрат по памяти, однако сильно уязвим к коллизиям. Но оценки по памяти и по времени остаются равны в среднем O(1).

3.7 Fixed Weighting

Метод балансировки с фиксированным весовым коэффициентом (Fixed Weighting [9]) - это алгоритм балансировки, при котором балансировщик присваивает вес каждому обработчику на основе выбранных им критериев для представления относительной способности каждого обработчика в системе обрабатывать поступающие пакеты. Обработчик с наибольшим весом будет получать все поступающие сегменты потока. Если обработчик с наибольшим весом выйдет из строя, все поступающие пакеты будут направлены на следующий обработчик с наибольшим весом.

К примеру трем обработчикам соответствуют веса 1, 2 и 3 - тогда все пакеты новых сегментов потока будут поступать на обработчик 3 до тех пор, пока он не выйдет из строя. После этого все пакеты начнут поступать на обработчик 2.

В реализации единственное, что придется помнить - таблица с весами для каждого обработчика, отсюда оценка по памяти - O(k).

С временной оценкой тоже все несложно, единственное, что возникает при балансировке - перебор по весам для каждого обработчика, отсюда оценка: O(k). А если изначально упорядочить обработчики по весам и хранить только индексы в массиве, то временная оценка будет O(1).

Однако следует отметить, что очевидно, что этот метод не достигнет необходимого равномерного распределения.

3.8 Выводы

Обзор содержит краткое представление о работе методов балансировки сегментов потока. Сравнение по перечисленным выше критериям представлены в таблице 1. Под критерием 'память' имеется ввиду память, затрачиваемая балансировщиком при работе с пакетами, под критерием 'время' имеется ввиду время балансировки, а под 'равнозначностью' имеется ввиду соблюдение равнозначности обработчиков. Последний критерий введен для проверки весовых методов - ведь в случае нашей задачи каждому обработчику будет присвоен одинаковый вес в связи с их одинаковыми вычислительными способностями, что переведет методы балансировки с весами в аналогичные им методы балансировки без весов. n - максимамальное возможное количество сегментов потока, для которых одновременно определены маршруты, k - общее количество обработчиков.

По результатам проведенного обзора для сравнения по эффективности были выбраны методы Round Robin, Least Connections, Resource Based и Hash Scheduling.

Критерии	Равнозначность	Память	Время[1]
RR	+	O(3n)	O(1)|O(n)
WRR	-	O(3n + k)	O(1)|O(n)
LC	+	O(3n+k)	O(k)|O(n+k)
WLC	-	O(3n+2k)	O(2k)|O(n+2k)
RB	+	O(3n+k)	O(2k)|O(n+2k)
HS	+	O(1)	O(1)
FW	-	O(k)	O(k)

Таблица 1: Сравнение методов балансировки

[1]разные оценки возникают за счет раличной реализации. Левый столбец - оценка без использования хэширования, правый - с использованием хэширования.

4 Описание стенда

4.1 Общее представление и цель использования стенда

Стенд представляет собой два сервера, соединенные напрямую: первый содержит DPDK, второй нужен для отправки пакетов для балансировки. Пакеты, отправленные со второго сервера на первый, поступают в кольцевой буфер DPDK, где балансируются по ядрам с помощью запущенной программы-балансировщика на первом сервере. Целью использования стенда является экспериментальное исследование по сравнению эффективности методов балансировки сегментов потока. Результаты балансировки выводятся на первом сервере для каждого из реализованных методов балансировки: Round Robin, Least Connections, Resource Based и Hash Scheduling.

4.2 Характеристики серверов

Сервер с DPDK:

- Кол-во ядер: 10
- Объём оперативной памяти: 64 Гб
- Объём дискового пространства: 256 Гб
- Операционная система: Ubuntu 22.04
- Версия ядра linux: 5.15.0-79-generic

Сервер для генерации пакетов:

- Кол-во ядер процессора: 2
- Объём оперативной памяти: 8 Гб
- Объём дискового пространства: 256 Гб
- Операционная система: Ubuntu 22.04
- Версия ядра linux: 5.15.0-82-generic

Машины соединены следующим образом (см. рис. 3): В самой работе активно использовалось соединение eno5—enp4s0f1, второе было добавлено для отказоустойчивости.

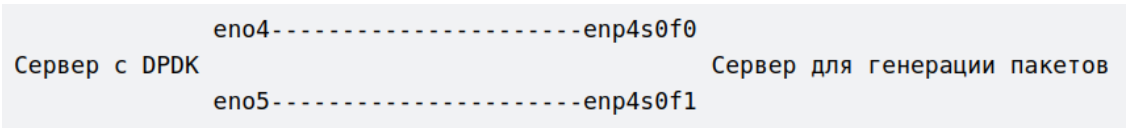

Рис. 3: Соединение серверов

4.3 Описание работы сервера с DPDK и программой-балансировщиком

Как было описано выше, посредством сетевых драйверов DPDK, пакеты проходят первичную обработку и попадают в кольцевой буфер системы DPDK. Первичная обработка заключается в сохранении пакетов в структуре rte_mbuf, а именно дозаполнение некоторых ее полей, включая поле hash.usr, использующееся в нашей задаче в качестве id сегмента потока.

В процессе работы балансировщика ядра системы соответствуют разным процессам, а именно:

1. RX процесс - процесс для получения пакетов и помещения их в кольцевой буфер на вход балансировщику.

2. Distributor процесс - процесс для балансировки пакетов. Решает, какому обработчику будет присвоен какой пакет, в зависимости от ID его сегмента. Приняв решение, записывает указатель на rte_mbuf пакета в выделенный буфер для отправки обработчику. Как только в этом буфере набирается определенное количество указателей, разом отправляет их рабочему ядру. При получении сообщений о законченной обработке с рабочих первого этапа (Request burst), отправляет указатели этих пакетов таким же образом на второй этап, а при получении сообщений о завершенной обработке со второго этапа направляет пакеты в кольцевой буфер на выходе, откуда позже их заберет следующий процесс.

3. TX процесс - процесс, забирающий пакеты из кольцевого буфера на выходе балансировщика и отправляющий их в нужные выходные порты.

4. Stats процесс - процесс, использующийся для вывода данных и работы консоли.

5. Worker процесс - процесс, использующийся в качестве обработчика пакетов.

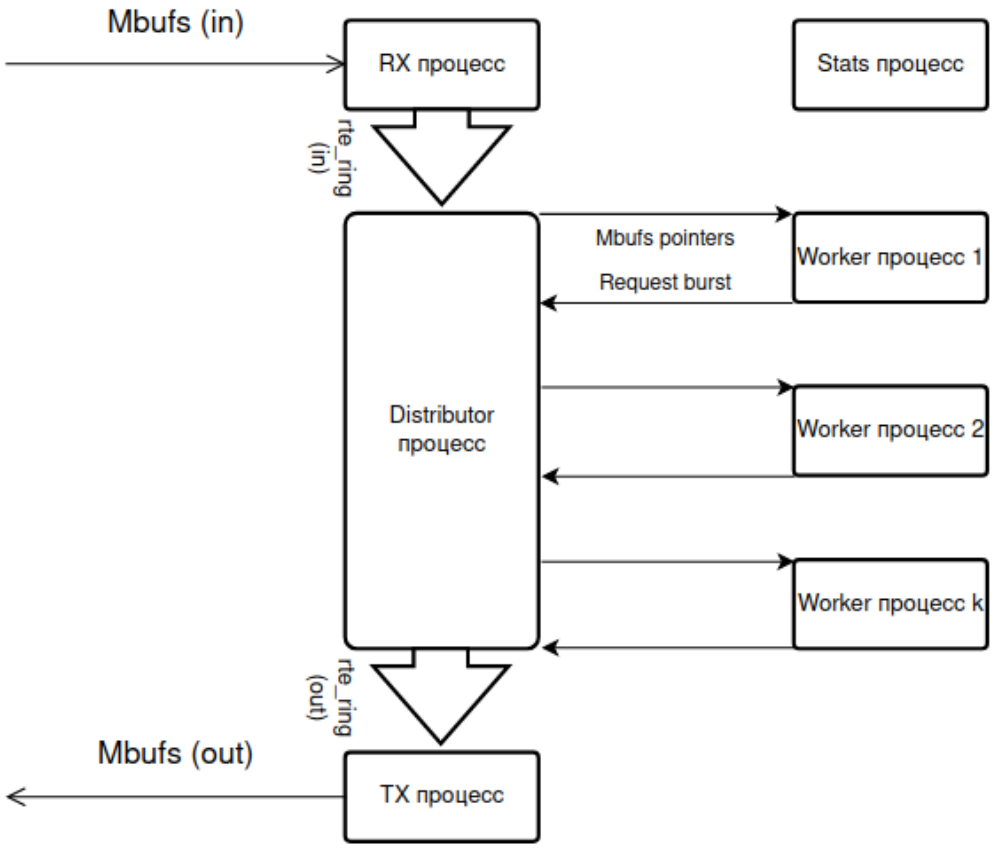

Рис. 4: Балансировщик [12]

На рисунке 4 можно увидеть взаимодействие ядер между собой в процессе осуществления балансировки.

Последовательная обработка на двух этапах достигается за счет повторного запуска функции rte_distributor_process(), которая отправляет пакеты, новоприбывшие в систему, на первый этап, а пакеты, возвращенные с рабочих ядер первого этапа, на следующий этап. На рисунке 5 можно увидеть более подробный путь, который проходит каждый пакет, попадая в ядро балансировщика.

Выбранные в третьем разделе методы балансировки были реализованы в различных функциях, вызываемых в функции rte_distributor_process(), запуск которой происходит на ядре балансировщика. Код функций и инструкции по запуску

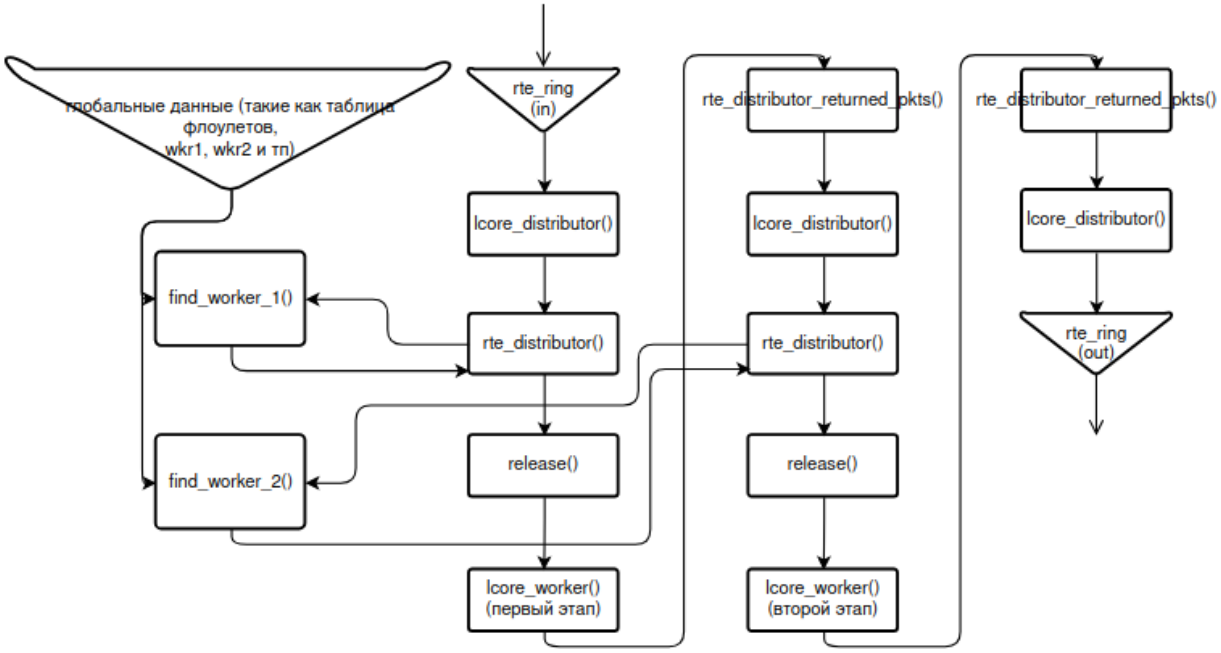

Рис. 5: Блок-схема с путем пакета в процессе балансировки

представлены в github[1].

4.4 Round Robin

Метод, требующий периодического обновления таблицы сегментов потоков. За обновлением таблицы следит lcore_distributor(), проверяя, не настало ли время обновить таблицу. Таблица содержит 4 поля:

1. Id сегмента потока
2. Номер обработчика с первого этапа
3. Номер обработчика со второго этапа
4. Время прибытия последнего пакета с этим ID сегмента потока

При обновлении таблицы, проверяется последнее поле для каждого сегмента потока. Если промежуток времени между последним прибытием и настоящим становится больше

[1]https://github.com/GoroshAD/load_balancing_dpdk

фиксированного δ, строка таблицы сбрасывается. Последняя ячейка таблицы, вообще говоря, не обязательна для функционирования метода балансировки и используется только для простого нахождения границ сегментов потоков.

Сам метод реализован в функциях find_worker1() и find_worker2(), в которых для каждого поступившего пакета в зависимости от ID сегмента выбирается обработчик соответсвенно первого и второго этапов обработки. Рассмотрим к примеру работу первой функции, вторая функция аналогична: при вызове функции сначала проверяется таблица сегментов, и если указанный ID уже есть в ней, возвращается номер обработчика, указанный в таблице. В случае нового ID таблица дозаписывается, а выбор номера обработчика происходит "по кругу" (в памяти хранится номер последнего записанного в таблицу обработчика, и этот номер обновляется при каждой записи). Функции возвращают номера нужных обработчиков по завершении.

Функция rte_distributor_process() получает номер нужного рабочего ядра и отправляет пакет в буфер на отправку обработчику. Как только буфер заполняется, вызывается функция release(), поставляющая заполненный буфер в очередь рабочему.

4.5 Least Connections

Таблица сегментов полностью аналогична представленной в предыдущем методе, а именно в методе Round Robin. Но в дополнение к этой таблице идут еще два массива:

1. Массив wkr1_connections содержит в себе количество сегментов потоков, проходящих через каждый обработчик первого этапа. Обновляется каждый раз, когда добавляется/сбрасывается сегмент потока в таблице сегментов. Обновление заключается в увеличении/уменьшении значения, соответствующего обработчику, через который будет проходить новый сегмент или проходил сброшенный.

2. Массив wkr2_connections содержит в себе количество сегментов потоков, проходщих через каждый обрботчик второго этапа. Обновление происходит аналогично массиву wkr2_connections.

Путь, преодолеваемый пакетом в процессе работы балансировщика остается тем же, меняются только функции, реализующие методы балансировки. В данном методе в процессе выбора номера рабочего ядра сравниваются значения в массивах каждого этапа, и возвращаемым номером обработчика является рабочий с наименьшим значением в массиве.

4.6 Resource Based

Последний из реализованных методов, использующих такую же таблицу сегментов. В этом методе в дополнение к таблице идут еще два массива, но уже отличных от метода Least Connections:

1. Массив wkr_table1 содержит в себе суммарное время обработки пакетов, прошедших через каждый обрботчик первого этапа.

2. Массив wkr_table2 содержит в себе суммарное время обработки пакетов, прошедших через каждый обрботчик второго этапа.

Путь, преодолеваемый пакетом в процессе работы балансировщика опять остается таким же и меняются только функции, реализующие методы балансировки. В этом случае в процессе выбора номера обработчика снова сравниваются значения в массивах каждого из этапов, а возвращаемым номеров обработчика является рабочее ядро с наименьшим значением в массиве (на этот раз возвращается обработчик с наименьшим суммарным временем работы).

4.7 Hash Scheduling

Данный метод балансировки уже не использует таблицы сегментов в предыдущем виде и дополнительных массивов со значениями для обработчиков. Все, что необходимо ему для балансировки - hash функция, вовзращающая номер обработчика в зависимости от id сегмента потока. Каждому этапу соответствует своя хэш-функция, вызываемая в find_worker1() и find_worker2().

Вообще говоря, в реализованном алгоритме flowlet_table() все еще присутствует, но используется в основном для динамического изменения констант для сегментов в хэш-функциях и не хранит в себе маршрутов сегментов потока.

4.8 Описание сервера для отправки пакетов

Основным средством по отправке пакетов служит python-библиотека scapy, специально предназначенная для работы с пакетами в сети. Запустив программу packet_sender.py, реализованную на этом сервере, пользователь сможет использовать следующие команды:

1. rnd_send NUM - команда, засылающая 256 пакетов с разными id сегментов потоков на первый сервер NUM раз (id встречаются повторно каждые 256 раз).

2. fix_send NUM - команда, отправляющая NUM пакетов с одинаковым id сегмента потока.

3. file_send PATH NUM - команда, отправляющая пакеты, содержащиеся в файле с путем PATH NUM раз.

4. testing PATH NUM - команда, запускающая экспериментальное исследование, то есть отправляющая пакеты из NUM файлов с пакетами, содержащиеся в директории PATH.

5 Экспериментальное исследование по сравнению эффективностей методов балансировки сегментов потока

5.1 Цель исследования

Необходимо сравнить эффективности методов балансировки сегментов потоков и определить метод, достигающий минимума максимального отклонения загруженности обработчиков от средней загруженности всех обработчиков. В качестве дополнительной цели проведем сравнение методов балансировки сегментов с методами балансировки потоков.

5.2 Методика проведения экспериментального исследования

Учитывая характеристики стенда и требования программы-балансировщика, на каждом этапе будут работать по 3 ядра-обработчика.

Данными для проведения экспериментов выбраны данные, содержащие обмен сообщениями между внутренней и внешней сетью кафедры АСВК факультета ВМК. Сбор данных о потоках сообщений производился одновременно из внешней и внутренней сети. В данном случае точкой сбора выступал программно-аппаратный комплекс на базе серверной платформы ASUS RS100-E4/PI2 под управлением операционной системы Linux (дистрибутив Debian Bookworm, версия ядра 6.1.0-11-686-pae). Комплекс изначально выполняет функции маршрутизатора и брандмауэра с отслеживанием состояния. Данные представлены в виде упорядоченной последовательности сетевых сообщений, представленных в виде байтовых строк, отправляемых и получаемых сетевыми интерфейсами маршрутизатора. Сообщения могут содержать информацию о метках виртуальных частный сетей канального уровня (VLAN, L2), IP адресах (IPv4, L3), использованных транспортных протоколах (TCP/UDP/SCTP, L4). Длительность процесса сбора данных – три календарных недели (21 день). Данные собраны с использованием утилиты tcpdump в режиме записи до 96 байт на каждое сообщение (с сохранением информации о количестве байт в исходном сообщении). Для экспериментов была выдана часть файлов.

Функциями для подсчета времени, в течение которого обработчик выполнял обработку пакетов, на первом этапе выбрана функция, возвращающая одинаковое время обработки пакетов, а на втором этапе функция возвращает разные значения для потоков. Сделано это было в связи с целью показать возможности методов на равных и на разных величинах времени обработки пакетов, так как не все методы учитывают эти величины в процессе балансировки.

Выходными данными являются файлы, в которые был перенаправлен вывод программы-балансировщика при отправки на первый сервер файлов со второго сервера. Файлы представляют собой сообщения балансировщика о состоянии системы и перечисления суммарного времени обработки пакетов каждого обработчика для каждого проведенного эксперимента. Теперь необходимо определить околичество проводимых экспериментов. Сначала требуется ввести случайную величину. Введем

$$\eta_{\gamma i_\gamma w} = |\xi_{i_\gamma w} - \overline{\xi_\gamma}|$$

По сути эта случайная величина характеризует отклонение загруженности обработчика от средней загруженности всех обработчиков его этапа за все эксперименты.

Далее необходимо определить доверительный интервал, в который попадает значение η (далее $\eta = \eta_{\gamma i_\gamma w}$) с вероятностью $p_0 = 0{,}95$. Сформулируем гипотезу H_0 : случайная величина попадает в доверительный интервал с вероятностью p_0. Теперь для заданной вероятности найдем необходимое число экспериментов для определения доверительного интервала. Для заданной вероятности $p_0 = 0,95$ число экспериментов составляет 442 (число экспериментов определялось аналогично [13]).

Результаты экспериментов

- Метод Round Robin (см. рис. 6)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа = [0.00470, 0.30531], второго - [0.00689, 0.34187]

- Метод Least Connections (см. рис. 7)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа равен [0.00282, 0.19577], второго - [0.00792, 0.29228]

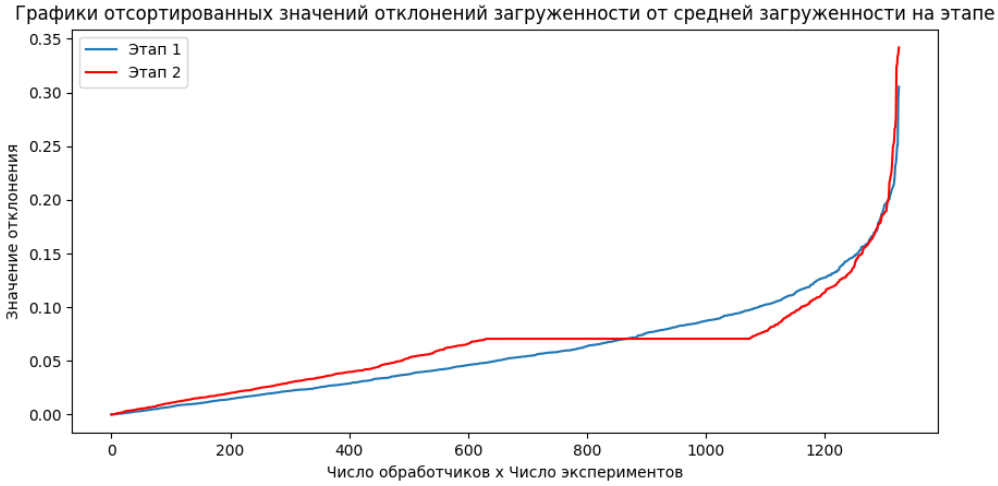

Рис. 6: Round Robin

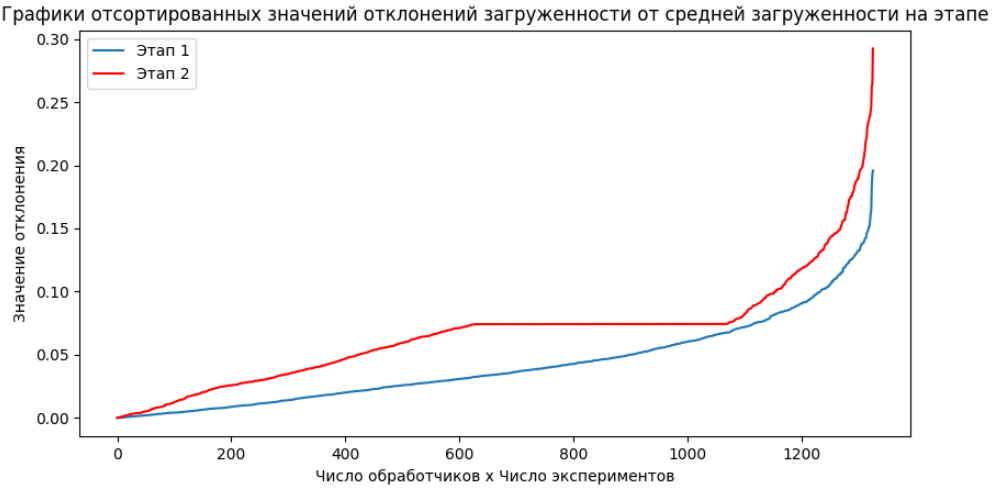

Рис. 7: Least Connections

- Метод Resource Based (см. рис. 8)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа равен [0.00099, 0.14719], второго - [0.00037, 0.02808]

- Метод Hash Scheduling (см. рис. 9)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа = [0.00074, 0.14086], второго = [0.00121, 0.06146]

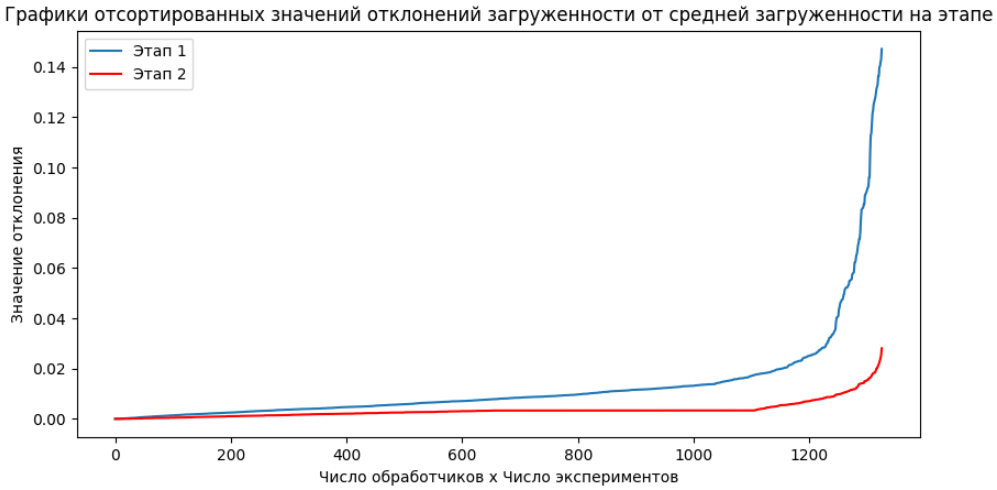

Рис. 8: Resource Based

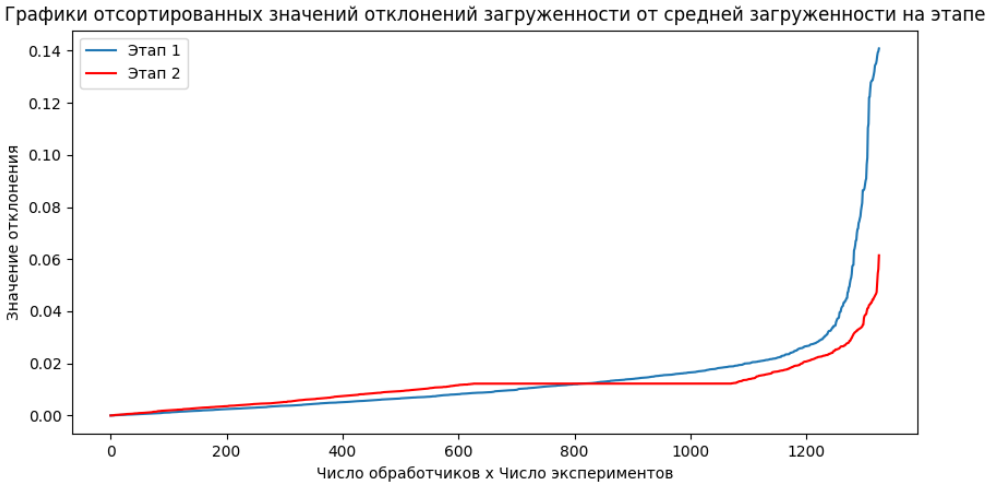

Рис. 9: Hash Scheduling

5.3 Анализ результатов

Используем полученные нами интервалы и вернемся к математической постановке. Необходимо отделить метод, который достигает минимума максимального значения случайной величины среди методов балансировки сегментов.

- Round Robin: $\max_{\gamma,i_\gamma,w}(\eta) = 0.34187$
- Least Connections: $\max_{\gamma,i_\gamma,w}(\eta) = 0.29228$
- Resource Based: $\max_{\gamma,i_\gamma,w}(\eta) = 0.14719$

- Hash Scheduling: $\max_{\gamma,i_\gamma,w}(\eta) = 0.14086$

Как видно из результатов, минимума достигает метод Hash Scheduling. Также следует отметить, что в реализации меньше всего ресурсов памяти использовалось именно этим методом. Время на балансировку также уходило меньше у этого метода. Но, если посмотреть значения на разных этапах обработки:

Первый этап:

- Round Robin: $\max_{\gamma,i_\gamma,w}(\eta) = 0.30531$
- Least Connections: $\max_{\gamma,i_\gamma,w}(\eta) = 0.19577$
- Resource Based: $\max_{\gamma,i_\gamma,w}(\eta) = 0.14719$
- Hash Scheduling: $\max_{\gamma,i_\gamma,w}(\eta) = 0.140858$

Второй этап:

- Round Robin: $\max_{\gamma,i_\gamma,w}(\eta) = 0.34187$
- Least Connections: $\max_{\gamma,i_\gamma,w}(\eta) = 0.29228$
- Resource Based: $\max_{\gamma,i_\gamma,w}(\eta) = 0.02808$
- Hash Scheduling: $\max_{\gamma,i_\gamma,w}(\eta) = 0.06146$

Видно, что на втором этапе, где согласно заданным входным данным разное время обработки пакетов, лидирует метод Resource Based. Причиной этого является то, что этот метод является единственным, учитывающим время обработки в выборе обработчика. Однако на первом этапе он уступает методу, основанному на хэшировании. Наилучшим методом по всем этапам обработки оказался Hash Scheduling.

5.4 Экспериментально сравнение по сравнению методов балансировки сегментов потоков с методами балансировки потоков

Используем те же данные для методов балансировки потоков и проведем то же количество экспериментов, а после сравним полученные результаты с результатми соответствующих им методов балансировки сегментов потоков.

Результаты экспериментов

- Метод Round Robin (см. рис. 10)

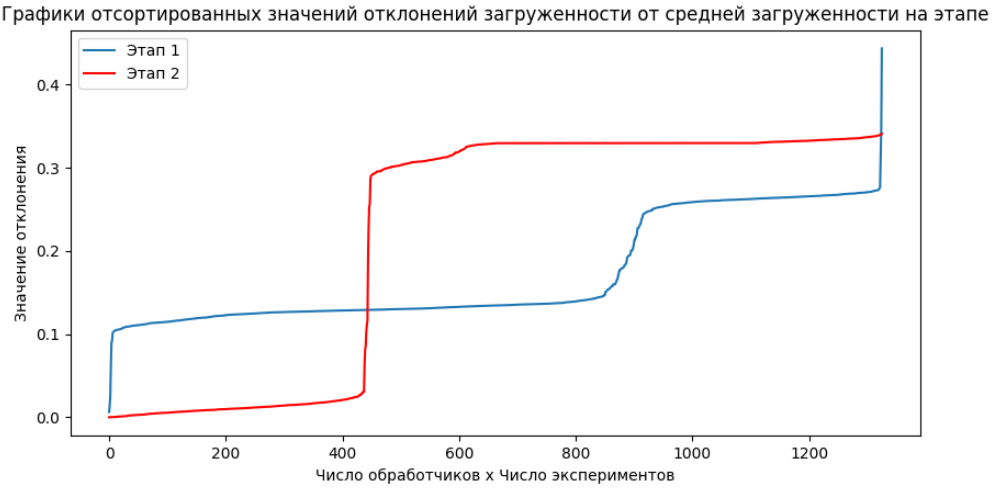

Рис. 10: Round Robin flows

  Соответственно, для вероятности 0.95, доверительный интервал первого этапа = [0.11261, 0.44362], второго = [0.00407, 0.34106]

- Метод Least Connections (см. рис. 11)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа равен [0.11454, 0.28425], второго - [0.00405, 0.34780]

- Метод Resource Based (см. рис. 12)
  Соответственно, для вероятности 0.95, доверительный интервал первого этапа равен [0.11261, 0.44362], второго - [0.00407, 0.34106]

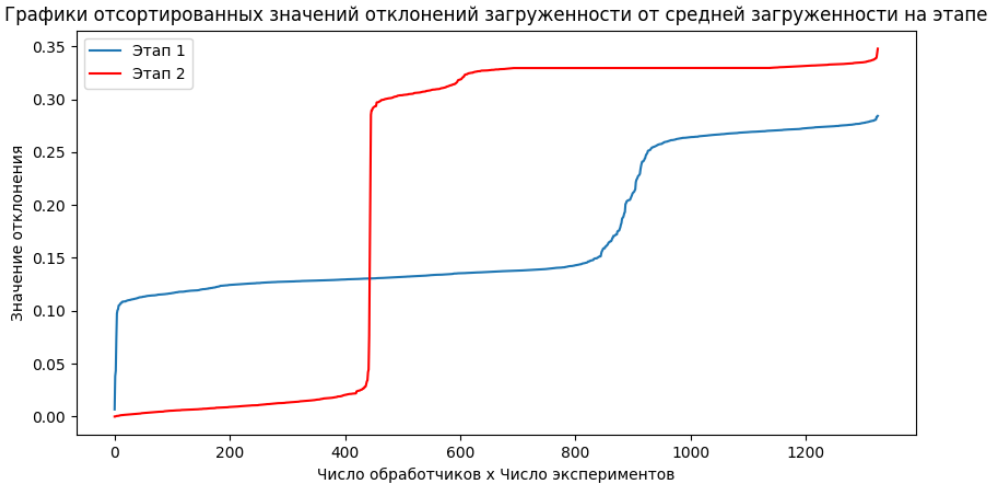

Рис. 11: Least Connections flows

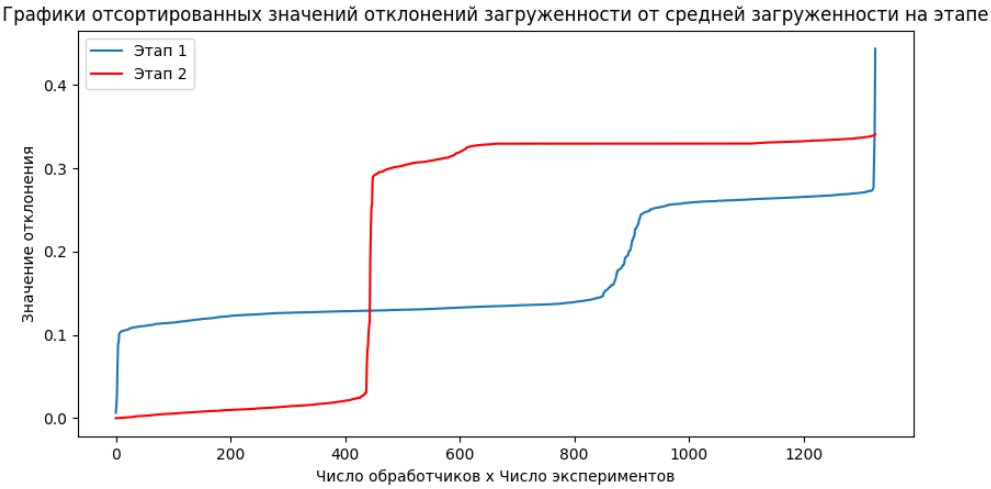

Рис. 12: Resource Based flows

- Метод Hash Scheduling

  Соответственно, для вероятности 0.95, доверительный интервал первого этапа = [0.17508, 0.45366], второго = [0.00200, 0.42552]

5.5 Анализ результатов

Используем полученные нами интервалы и определим максимально отклонение загруженности каждого метода:

- Round Robin: $\max_{\gamma, i_\gamma, w}(\eta) = 0.44362$

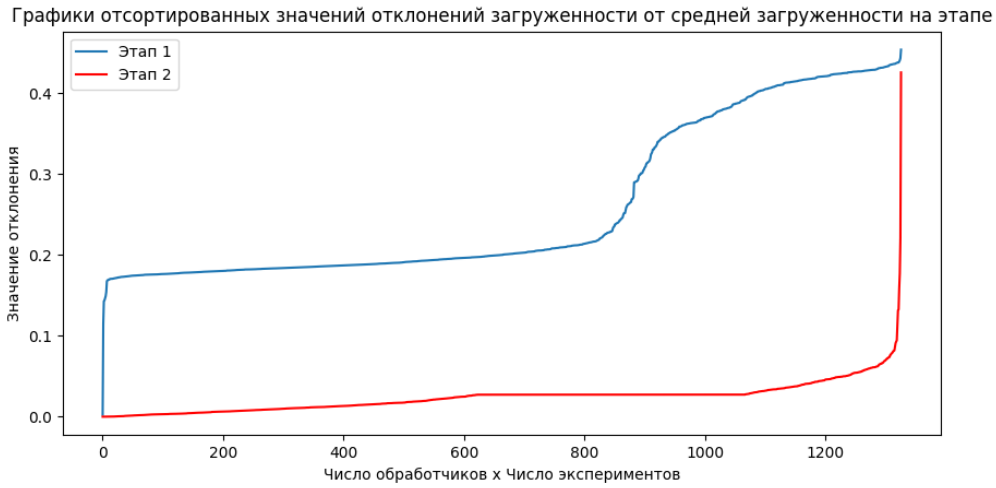

Рис. 13: Hash Scheduling flows

- Least Connections: $\max_{\gamma,i_\gamma,w}(\eta) = 0.34780$
- Resource Based: $\max_{\gamma,i_\gamma,w}(\eta) = 0.44362$
- Hash Scheduling: $\max_{\gamma,i_\gamma,w}(\eta) = 0.45366$

Сравним полученные значения методов балансировки потоков со значениями, полученными в предыдущем исследовании (см. таблицу 2).

Метод балансировки	сегменты потока	потоки
Round Robin	0.34187	0.44362
Least Connections	0.29228	0.34780
Resource Based	0.14719	0.44362
Hash Scheduling	0.14086	0.45366

Таблица 2: Сравнение результатов методов балансировки сегментов и потоков

Видно, что каждый метод балансировки сегментов потоков получил значения меньше, чем соответствующий ему метод балансировки потоков.

6 Заключение

В ходе выполнения настоящей работы были получены следующие результаты:

- Сделана математическая постановка задачи балансировки сегментов потока по ядрам двухэтапного конвейера обработки пакетов DPDK.
- По результатам проведенного обзора методов балансировки, для реализации и сравнения эффективностей было выбрано 4 метода: Round Robin, Least Connections, Resource Based и Hash Scheduling;
- Разработаны и реализованы алгоритмы балансировки сегментов потока на основе выбранных методов.
- Исследована эффективность распределения сегментов потока по ядрам для реализованных алгоритмов балансировки, и по результатам исследования метод Hash Scheduling показал наименьшее значение отклонения загруженности, а также было показано улучшение этого показателя по сравнению с балансировкой потоками.

Дальнейший интерес для исследования представляют:

- Масштабирование экспериментов - увеличение количества обработчиков, вариантов распределения обработчиков по этапам и так далее.
- Исследование и проведение сравнения методов балансировки сегментов, использующих машинное обучение.

Список литературы

[1] Global Internet traffic growth forecast: Looking forward from 2024. — https://edgeoptic.com/global-internet-traffic-growth-forecast-looking-forward-from-2024/ — [Online; accessed 27-May-2024].

[2] *Van Jacobson, Michael J. Karels.* Congestion Avoidance and Control / Michael J. Karels Van Jacobson // SIGCOMM '88: Symposium proceedings on Communications architectures and protocols. — 1988. — Pp. 314–329.

[3] *Xinglong Diao Huaxi Gu, Xiaoshan Yu Liang Qin Changyun Luo.* Flex: A flowlet-level load balancing based on load-adaptive timeout in DCN / Xiaoshan Yu Liang Qin Changyun Luo Xinglong Diao, Huaxi Gu // Future Generation Computer Systems. — 2022. — Pp. 219–230.

[4] DPDK documentation. — https://doc.dpdk.org/guides/index.html. — [Online; accessed 25-May-2024].

[5] Linux Kernel Docs. — https://docs.kernel.org/networking/index.html. — [Online; accessed 26-May-2024].

[6] *Yemelianov, Andrei.* Введение в DPDK: архитектура и принцип работы. — https://habr.com/ru/companies/selectel/articles/313150/. — 2016. — [Online; accessed 26-May-2024].

[7] *Divyansh Singh Vandit Bhalla, Neha Garg.* Load Balancing Algorithms with the Application of Machine Learning: A review. / Neha Garg Divyansh Singh, Vandit Bhalla // MR International Journal of Engineering and Technology. — Vol. 10. — 2023.

[8] *Markova, Vasilena.* Round Robin Load Balancing. Simple and efficient. — https://www.cloudns.net/blog/round-robin-load-balancing/. — 2023. — [Online; accessed 26-May-2024].

[9] *Lütkebohle, Ingo.* Kemp Load Balancing Algorithms and Techniques. — https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques. — [Online; accessed 25-May-2024].

[10] Load Balancing Algorithm Explained. — https://aws.amazon.com/ru/what-is/load-balancing/. — [Online; accessed 26-May-2024].

[11] Least connection method, Hashing methods. — https://docs.netscaler.com/en-us/citrix-adc/current-release/load-balancing/load-balancing-customizing-algorithms/. — 2024. — [Online; accessed 26-May-2024].

[12] Distributor Sample Application. — https://dpdk-power-docs.readthedocs.io/en/latest/sample_app_ug/dist_app.html. — [Online; accessed 25-May-2024].

[13] *Balashov, V. V.* Ensuring compatibility of requirements to the schedule of exchange over a channel with centralized control. / V. V. Balashov. — 2011.

[14] *Benet C. H., Kassler A. J.* Flowdyn: Towards a dynamic flowlet gap detection using programmable data planes / Kassler A. J. Benet C. H. // 2019 IEEE 8th International Conference on Cloud Networking (CloudNet). — 2019. — Pp. 1–7.

[15] *Sinha S. Kandula S., Katabi D.* Harnessing TCP's burstiness with flowlet switching / Katabi D. Sinha S., Kandula S. // Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III). — Citeseer, 2004.

[16] *Li, Y.* FlowRadar: A Better NetFlow for Data Centers / Y. Li // 13th USENIX symposium on networked systems design and implementation (NSDI 16). — 2016. — Pp. 311–324.

[17] *Trick, Christopher*. What is DPDK (Data Plane Development Kit)? — https://www.trentonsystems.com/en-us/resource-hub/blog/what-is-dpdk. — [Online; accessed 25-May-2024].