Московский государственный университет
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Стрельников Алексей Олегович

Выбор наилучшего алгоритма кэширования для заданных реальных данных

Выпускная квалификационная работа

Научный руководитель:
математик ЛВК
А. М. Колосов

Москва, 2022

Аннотация

В работе проводится обзор стратегий кэширования, а также для заданного набора алгоритмов кэширования и заданного набора реальных данных строится отображение, позволяющее определить наилучший по заданному критерию алгоритм кэширования для указанных данных, что позволяет выбирать наилучший алгоритм из заданных. Кроме того, предлагается подход для выбора наилучшего алгоритма кэширования без запуска на реальных данных и проводится экспериментальное исследование, обосновывающее применимость подобного подхода.

Содержание

1 Введение

Алгоритм — одно из основных понятий в вычислительной математике. Различные алгоритмы характеризуются некоторыми *показателями качества*.

Для ускорения работы разнообразных систем используется *кэширование*. Кэширование основывается на хранении в промежуточном буфере быстрого доступа (в *кэше*) некоторых подходящих объектов, например, мультимедиа данных или инструкций процессора. Выбор этих объектов для помещения в кэш (и выбор объектов для удаления из него) основывается на алгоритмах, называемых *стратегиями кэширования*, причём показатель качества для них зависит как от стратегии, так и от поступающих на вход данных.

Введение показателя качества позволяет сравнивать стратегии кэширования на заранее заданном наборе данных. Зачастую используется **OHR** (*object hit rate*) — отношение количества кэш-попаданий к количеству всех запросов. Однако подобный показатель может быть неэффективен, когда размеры объектов значительно отличаются. В таком случае эффективнее использовать показатель **BHR** (*byte hit rate* — показатель попадания байтов) — отношение размера всех объектов, находившихся в кэше во время запроса, к сумме размеров всех запрошенных объектов.

2 Используемая терминология

2.1 Исходные данные

В качестве исходных данных используются наборы запросов к реальным данным (запросов к видеохостингу) суммарным объёмом более 10 Гб. Запросы поступают к одному и тому же видеохостингу и локализованы по местности.

Сами запросы описываются таблицей, состоящей из 9 столбцов. Фрагмент такой таблицы изображён ниже.

Время запроса	Тип сервиса	PoP id	ID объекта	Длительность контента (с)	Размер объекта (Кб)	Bitrate объекта (Кб/с)	Запрашиваемый размер	Длительность запроса
1483200009	16	79	269780084	116.0	23359.0	1611.0	0.0	115.0
1483200009	64	167	271448489	1999.0	992503.0	3972.0	1727275.0	738.0
1483200009	64	79	269684392	1413.0	163731.0	927.0	432853.0	871.0
1483200009	64	139	271309855	2700.0	417487.0	2352.602	420967.0	2240.0

Таблица 1: Фрагмент потока запросов к видеохостингу

Однако указанный поток запросов, во-первых, содержит некорректные данные (например, существуют строчки, где значение в графе "запрашиваемый размер"равны нулю) и, во-вторых, не все из этих данных используются в ходе кэширования.

По этой причине было решено использовать таблицу из трёх столбцов, которая содержит всю необходимую для работы алгоритмов кэширования информацию. Если преобразовать фрагмент изначального потока запросов, то будет получена следующая таблица.

Время запроса	ID объекта	Размер объекта (Кб)
1483200009	269780084	23359.0
1483200009	271448489	992503.0
1483200009	269684392	163731.0
1483200009	271309855	417487.0

Таблица 2: Фрагмент преобразованного потока запросов к видеохостингу

2.2 Информационные объекты

Формализация начинается с введения понятия **информационного объекта** ω. Пусть Ω - множество всех информационных объектов.

Пусть задано некоторое **множество идентификаторов** $\mathbb{I}$. Тогда $I \in \mathbb{I}$ - идентификатор. Идентификатор, соответствующий объекту ω, обозначается как I_ω, например, $I_\omega = 00003055812$. Множество идентификаторов задаётся *почти* произвольным: накладывается ограничение уникальности идентификаторов для каждого объекта: $\forall \omega_1, \omega_2 \in \Omega : \omega_1 \neq \omega_2 \Rightarrow I_{\omega_1} \neq I_{\omega_2}$.

Тогда **информационный объект** ω определяется как упорядоченная пара, состоящая из идентификатора и размера объекта: $\omega = \{I_\omega, R_\omega\}$, где $R_\omega \in \mathbb{R}, R_\omega > 0$. Например,

$$\omega' = \{00003055812, 2G\}, \omega* = \{000019241511, 500M\}$$

В работе сама структура информационного объекта (кроме размера) не имеет значения, поскольку кэш не хранит об этом никакой информации.

Стоит отметить, что существует сюръективное отображение из множества идентификаторов в $\mathbb{R}$: каждому идентификатору соответствует некоторый размер объекта.

$$\exists g : \forall I \in \mathbb{I}\ \exists R \in \mathbb{R} : g(I) = R$$

Восстановить же идентификатор по размеру бывает невозможно: так, одному и тому же размеру могут соответствовать несколько идентификаторов или же не соответствовать ни одного.

2.3 Запросы и трассы

Вводится понятие **трассы**. Для этого необходимо определить **запрос**. Под запросом Z понимается тройка вида (I_ω, R_ω, t), где $\omega \in \Omega' \subseteq \Omega$, а $t \in \mathbb{N} \cup \{0\}$ - некоторый момент времени. Тогда трасса длины n определяется как конечная последовательность запросов: $T = (Z_1, Z_2, ..., Z_n)$. Длина трассы будет обозначаться как $|T| = n$. Пример трассы:

$$bcdc3bd126, 25159, 3633$$
$$e5304ac9f7, 78623, 3637$$

Однако не все подобные последовательности считаются корректными. Важным ограничением, накладываемым на **корректную трассу**, является однозначное соответствие размера идентификатору. Иными словами, для корректной трассы должно выполняться условие:

$$\forall Z_1 = (I_1, R_1, t_1), Z_2 = (I_2, R_2, t_2) \in T : I_1 = I_2 \Rightarrow R_1 = R_2$$

.

Например, некорректной считается следующая трасса длины 2:

$$bcdc3bd126, 25159, 3630$$
$$bcdc3bd126, 26324, 3631$$

Стоит отметить, что в один момент времени могут запрашиваться несколько объектов, причём, возможно, совпадающих. Так, последовательность запросов

$$bcdc3bd126, 25159, 3630$$
$$e5304ac9f7, 78623, 3630$$
$$bcdc3bd126, 25159, 3630$$

считается корректной трассой длины 3.

В качестве данных, для которых выбирается алгоритм кэширования, в работе рассматриваются именно трассы. В дальнейшем эти понятия считаются синонимами и отождествляются.

2.4 Сравнение трасс

Для сравнения алгоритмов кэширования между собой необходимо также иметь возможность сравнить и поступающие им на вход данные, получая конечное число – схожесть между трассами. Иными словами, необходимо ввести *функцию сходства*.

Сравнивать трассы в том виде, в каком они были введены, проблематично. По этой причине рассматривается семейство функций
$\mathbb{F} = \{f_1, f_2, ..., f_k, ...\} : \forall i \ f_i : \mathbb{T} \to \mathbb{R}^{m_i}$, где $\mathbb{T}$ – множество всех корректных трасс, а $m_i \in \mathbb{N}$ - число, определяемое функцией f_i. Иначе говоря, изучаются отображения трасс в векторы. Вектор v, полученный из трассы T воздействием на неё функцией f, обозначается v_f^T.

Необходимо также выбрать метрику для векторов. По аналогии рассматриваются функции $\mathbb{P} = \{\rho_1, \rho_2, ..., \rho_q, ...\} : \forall j \ \rho_j : (\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}$. Тогда, выбрав некоторую функцию $f \in \mathbb{F}$, а также $\rho \in \mathbb{P}$, определяется сходство между трассами T_i и T_j как $\rho_f(T_i, T_j) = \rho(f(T_i), f(T_j)) = \rho(v_f^{T_i}, v_f^{T_j})$.

Считается, что трасса T_j ближе к T_i, чем T_k, в смысле функции ρ в контексте f, если $\rho_f(T_i, T_j) < \rho_f(T_i, T_k)$.

Вводится рекурсивное определение k-го ближайшего соседа в смысле функции ρ в контексте f (k-ую ближайшую трассу), $k \in \mathbb{N} \cup \{0\}$, к трассе T_i. Считается, что *нулевым ближайшим соседом* к трассе T_i является она сама (здесь функции ρ и f роли не играют). Предполагается, что все трассы различны, то есть различаются хотя бы одним запросом из потока (в противном случае такие трассы отождествляются).

Пусть определены ближайшие соседи в смысле функции ρ в контексте f для трассы T_i вплоть до k-го, $k \in \mathbb{N} \cup \{0\}$. Множество таких соседей будет обозначаться

как $N_k^{\rho_f} = \{T_i, T_{c_1}, ..., T_{c_t}\}$, где $t \geq k$ (возможна такая ситуация, где несколько разных трасс равноудалены от T_i).

Тогда k+1-ым ближайшим соседом (k+1-ой ближайшей трассой) к T_i в смысле функции ρ в контексте f будут считаться все трассы T_j :
$\rho_f(T_i, T_j) = \min\limits_{\forall T \in \mathbb{T} \setminus N_k^{\rho_f}} \rho_f(T_i, T)$.

В частности, ближайшей трассой (*первым ближайшим соседом*, или далее просто *ближайшим соседом*) к T_i в смысле функции ρ в контексте f называются все трассы $T_j : \rho_f(T_i, T_j) = \min\limits_{\forall T \in \mathbb{T} \setminus \{T_i\}} \rho_f(T_i, T)$.

2.5 Качество алгоритма на трассе

Помимо этого, важно также учитывать функцию измерения качества работы алгоритма alg на трассе T. Пусть $\mathfrak{Q} = \{Q_1, Q_2, ..., Q_l, ...\}$ – семейство таких функций. Следовательно при выбранном и зафиксированном $Q \in \mathfrak{Q} : (Alg, \mathbb{T}) \to \mathbb{R}$, где Alg – множество алгоритмов, показатель качества для алгоритма alg на трассе T определяется как значение $Q(alg, T)$.

В зависимости от выбранной функции Q, очевидно, зависит и результат определения **наилучшего алгоритма** на заданной трассе, то есть такого алгоритма, значение функции Q для которого будет максимально. Иначе говоря, алгоритм alg^* считается наилучшим на трассе T, если $Q(alg^*, T) = \max\limits_{alg \in Alg} Q(alg, T)$.

3 Постановка и актуальность задачи

В введённых обозначениях формулируется постановка задачи.

3.1 Постановка задачи

Необходимо для заданных алгоритмов кэширования $Alg = \{alg_1, ..., alg_n\}$ провести построение и оценку качества классификатора трасс $T \in \mathbb{T}$. Формально это означает, что необходимо:

1. Выбрать алгоритмы кэширования $Alg^* = \{alg_1, alg_2, ..., alg_m\} \in Alg$;

2. Выбрать подмножество трасс $\mathbb{T}^* \subset \mathbb{T}$, на которых будет проверяться качество алгоритмов, а также провести разделение на обучающую $\mathbb{T}_{train}$ и тренировочную $\mathbb{T}_{test}$ выборки;

3. Выбрать функцию качества Q и вычислить значения $Q(alg_i, T_j)\ \forall i, j$: $alg_i \in Alg^*, T_j \in \mathbb{T}_{train}$;

4. Определить функцию преобразования трасс в векторы f, а также набор функций сходства $\mathbb{P}^* = \{\rho_1, \rho_2, ..., \rho_q\}$;

5. На основании полученных данных разработать классификатор, определяющий наилучший алгоритм кэширования для заданной трассы или, иначе говоря, предсказывающий k:
   $Q(alg_k, T_i) = \max\limits_{1 \le j \le m} Q(alg_j, T_i)\ \forall i : T_i \in \mathbb{T}^*$.

   Фактически необходимо найти отображение q: $\mathbb{T}^* \to Alg^*$:
   $\forall T \in \mathbb{T}\ Q(q(T), T) = \max\limits_{alg \in Alg^*} Q(alg, T)$ (классификатор и будет этим отображением).

   Для проверки корректности построенного отображения q^* критерием будет доля трасс, которым сопоставлен алгоритм кэширования с наибольшим показателем качества. Иначе говоря, корректность определяется значением выражения

   $$\frac{\sum\limits_{T \in \mathbb{T}^*} I_{\{q^*(T)=q(T)\}}}{|\mathbb{T}^*|},$$

   где q^* – построенное, а q – истинное отображения,
   $I_{\{q^*(T)=q(T)\}} = \begin{cases} 1, & \text{если } q^*(T) = q(T), \\ 0, & \text{иначе} \end{cases}$

3.2 План работы

План работы описывается следующей структурой:

1. **Выбор** формата представления данных;
2. **Выбор** различных по принципу своей работы алгоритмов кэширования и разметка данных;
3. **Разработка** программного комплекса, необходимого для проведения экспериментов;
4. **Разработка** классификатора трасс;
5. **Проведение** экспериментов для оценки качества классификации.

3.3 Актуальность работы

Зная результаты для достаточно малого набора потоков запросов, предсказывается алгоритм, для которого будет получен наибольший показатель качества на другой трассе. Тем самым время, затрачиваемое на непосредственный запуск алгоритмов на трассах, для того, чтобы определить показатель качества алгоритмов на них, экономится. В особенности подобный классификатор полезен, если значения функции $Q(alg, T)$ вычисляются длительное время (например, если трасса T имеет длину порядка 10^9): от нескольких часов до нескольких дней.

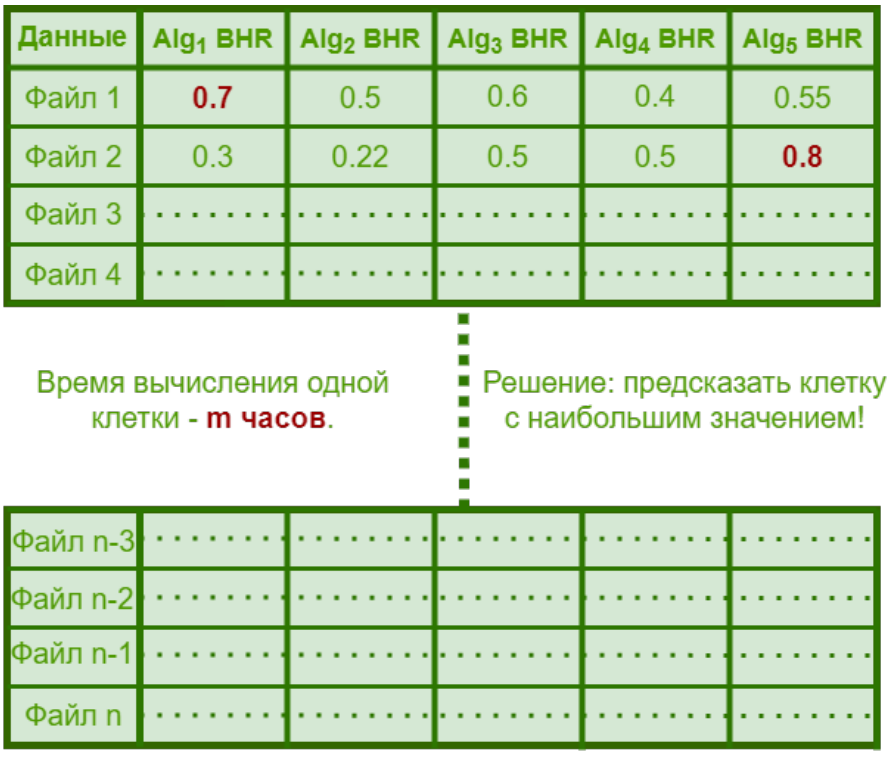

Рис. 1: *Пример таблицы показателей качества для алгоритмов кэширования*

Актуальность работы иллюстрируется рисунком 1.

Считается, что каждая клетка в данной таблице вычисляется по меньшей мере 1 час, и при этом имеется 5 алгоритмов кэширования. Тогда заполнение всех клеток таблицы займёт $5n$ часов. Как правило, количество файлов n велико (порядка 10^3, 10^4 или больше), однако уже для 10^3 файлов вычисление всей таблицы занимает более 208 дней. Именно по этой причине возникает задача выбора алгоритма кэширования без вычисления значения функции качества.

4 Обзор стратегий кэширования

Одним из основных понятий в работе является понятие **информационного объекта** — множества, которое содержит в себе характеристики объекта (в частности, уникальный для объекта идентификатор, а также **размер** объекта — положительное число). Именно идентификатор и размер используются при запросе объекта.

В каждый *момент времени* кэш содержит в себе некоторое множество объектов, суммарный размер которых не превышает *размер кэша*. При получении нового объекта в зависимости от содержания кэша осуществляются следующие действия:

- Если объект находился в кэше (случай *кэш-попадания*), то тогда содержание кэша не меняется, а запрашиваемый объект возвращается;
- Если объекта в кэше не было (*кэш-промах*), то, согласно стратегии кэширования, объект может быть добавлен в кэш, возможно, с удалением одного или нескольких объектов из него.

Точная последовательности запросов информационных объектов необходима для моделирования работа кэша **симулятором** — программой, принимающей на вход *трассу*, являющуюся историей запросов, которая затем обрабатывается согласно стратегии кэширования. Результатом работы симулятора является файл, содержащий в себе информацию об изменении показателя качества по ходу работы алгоритма.

4.1 Простейшие стратегии: LIFO, FIFO, LRU, LFU

Существует ряд простых в реализации стратегий кэширования. В частности, к ним относятся:

- **LIFO** (*Last In - First Out* — Последний пришёл - первый ушёл). Из кэша удаляется объект, пришедший последним. По такому принципу организован стек.

- **FIFO** (*First In - First Out* — Первый пришёл - первый ушёл). Из кэша удаляется наиболее "старый"объект. Например, если кэш вмещает 2 объекта, то при последовательности запросов 1, 2, 3, 2, 4 в кэше будут находиться объекты {1}, {1, 2}, {2, 3}, {3, 2}, {2, 4}. Подобный способ работы с данными используется в очереди.

- **LRU** (*Least Recently Used* — наименее недавно используемый). Из кэша удаляется объект, который не запрашивался дольше всех остальных по времени.

- **LFU** (*Least Frequently Used* — наименее часто используемый). Из кэша удаляется объект, который запрашивался реже остальных по количеству обращений.

Подобные стратегии, как правило, не слишком эффективны ввиду своей простоты и нацеленности на определённый класс данных. При этом в подобных стратегиях запрошенный объект, который не находился в кэш-памяти, всегда будет добавляться. Обычно такие алгоритмы используются в качестве основы для построения более сложных стратегий.

4.2 Модификации простейших стратегий: SNLRU, MQ и другие

Как было указано, для улучшения простейших стратегий используют достаточно простые модификации, позволяющие добиться большей эффективности. Подобные модификации наиболее часто связаны с разделением кэша на части — *сегменты* Примерами могут служить **SNLRU** (*N-Segmented LRU*), **MidPointLRU**, **2Q** (*2-Queue*), **MQ** (*Multi-Queue*).

Сегменты могут работать как независимо друг от друга (т. е. фактически кэш представляет объединение нескольких независимых, более маленьких кэшей), так и зависимо от остальных — в таком случае объекты перемещаются между сегментами согласно некоторому принципу.

Так, при стратегии **SNLRU** кэш делится на N сегментов, каждый из которых представляет собой **LRU**-кэш. Изначально объекты попадают в последний, "нижний"сегмент, но при увеличении количества запросов к объекту он перемещается в следующий, более "высокий" сегмент. Соответственно, если объект долго

не запрашивается, то он перемещается в более "низкий" сегмент. Если же объект находился в последнем сегменте, то он удаляется из кэша.

MidPointLRU фактически является вариацией этого алгоритма с 2 сегментами.

Стратегия **2Q** основывается на идее разделения кэша на 3 сегмента: двое из них представляют собой **FIFO**-кэши (условно назовём их *In* и *Out*), а последний — **LRU**. Изначально все новые объекты попадают в кэш *In*. Далее стратегия зависит от сегмента, в котором находится объект:

- **In**: при удалении объекта он перемещается в *Out*, при запросе ничего не происходит;
- **Out**: при запросе объкта он перемещается в *LRU*, при удалении удаляется из кэша окончательно;
- **LRU**: работает как стандартный *LRU*.

Обобщением данной стратегии является стратегия **MQ** [1], в которой используется больше сегментов.

Подобные стратегии уже учитывают частоту запроса элементов: это позволяет избежать удаления популярных элементов, обращение к которым идёт достаточно часто, но которые периодически всё равно удаляются из кэш-памяти (что является одним из недостатков *LRU*). Впрочем, такие алгоритмы также могут оказаться не слишком эффективными, если запрашиваемые объекты будут иметь достаточно большие размеры: объём каждого сегмента меньше, чем у всего кэша, из-за чего запрос объёмного непопулярного объекта может привести к тому, что сегмент выбросит из памяти много популярных, но менее объёмных объектов.

Кроме того, существуют и иные модификации простейших стратегий. Так, например, известен алгоритм **ARC** *(Adaptive Replacement Cache)* [2], который с помощью истории запросов адаптивно определяет, какая из стратегий (*LRU* или *LFU*) более предпочтительна в данный момент времени, и меняет размер соответствующих сегментов.

4.3 Иные стратегии: Clock, Scache.

Существует ряд стратегий, которые основываются на иных идеях.

Так, известен алгоритм **CLOCK** *("часы")*, где у каждого объекта имеется флаг. Изначально он находится в состоянии 0. При запросе объекта этот флаг устанавливается равным 1. Если необходимо вытеснить какой-то из объектов, то их список просматривается, начиная с позиции элемента, следующего за последним удалённым, причём считается, что за последним элементом следует 1-ый (т.е. если ранее удалялся элемент под номером 3, то поиск начнётся либо с начала списка, если его размер равен 3, либо же с 4-ой позиции). Если флаг объекта равен 1,

то он устанавливается равным 0; если же флаг нулевой, то объект вытесняется. При вытеснении какого-либо элемента поиск останавливается.

Подобный алгоритм прост в реализации, но при этом может давать хорошие результаты.

Существует также ряд модификаций данного алгоритма (**GCLOCK** - *Generalized Clock*, **Clock-Pro**, **CAR** - *Clock with Adaptive Replacement*) [3], основанных на ином изменении флагов.

Для запросов, имеющих некоторое периодическое распределение, был предложен **Scache** — *атрибутивно-сегментный кэш*. Подробно он описан в [4]. Кэш динамически делится на независимые сегменты, каждому из которых соответствует некоторое подмножество из всех объектов, не пересекающееся с другими подмножествами.

4.4 Оптимальная стратегия

Оптимальную стратегию иначе также называют *стратегией с предвидением, алгоритмом Белади* [5]. Подобная стратегия является модельной абстракцией, поскольку её реализация невозможна: кэшу должны быть известны не только поступающие запросы, но и те, что произойдут в будущем. Подобная стратегия выступает в качестве верхней оценки эффективности других стратегий кэширования. Стратегия состоит в том, что при необходимости удаления объекта будет "выброшен" тот, который не будет запрошен в будущем дольше всего.

Известны также "приближения" алгоритма Белади, которые базируются на уже имеющейся истории запросов. Так, например, существует алгоритм *Hawkeye*, описанный в [6].

4.5 Классы алгоритмов

Для проведения обзора предлагается ввести понятие *классов алгоритмов* – конечного подмножества алгоритмов, объединённых по принципу работы. Так, например, выделяется класс алгоритмов, основанных на LRU, к которому относится сам кэш LRU и его сегментированные модификации (*SNLRU*, *MidPoint LRU* и иные). Введение такого разделения обосновывается далее.

Для двух алгоритмов определяется "зазор". Рассматривается фиксированное множество трасс $\mathbb{T}$. Тогда зазор $g(alg_1, alg_2)$ для двух алгоритмов alg_1 и alg_2 и множества $\mathbb{T}$ определяется как максимум модуля разности показателей качества, соответствующих указанным алгоритмам, для трасс из $\mathbb{T}$. Иначе говоря,
$g(alg_1, alg_2) = \max\limits_{T \in \mathbb{T}} |Q(alg_1, T) - Q(alg_2, T)|$. Считается, что функция качества Q выбрана и зафиксирована.

Пусть $Alg^* = \{alg_1^*, alg_2^*, ..., alg_k^*\}$ – некоторый класс алгоритмов. Пусть также имеется алгоритм alg' (возможно, принадлежащий классу алгоритмов Alg^*).

Важно заметить, что, вообще говоря, $g(alg', alg_i) \neq g(alg', alg_j), i \neq j, 1 \leq i, j \leq k$. Для определённости зазор между классом алгоритмов Alg^* и алгоритмом alg' вводится как максимум среди всех зазоров для алгоритма alg' и для алгоритмов из Alg^*, то есть $g(Alg^*, alg') = \max\limits_{alg^* \in Alg^*} g(alg^*, alg')$.

Зазор также определяется и для двух классов алгоритмов Alg_1, Alg_2. Тогда $g(Alg_1, Alg_2) = \max\limits_{alg_1 \in Alg_1, alg_2 \in Alg_2} g(alg_1, alg_2)$.

Разделение алгоритмов на классы позволяет выбрать и зафиксировать для двух классов Alg_1 и Alg_2 те алгоритмы alg_1 и alg_2, на которых достигается максимум функции g. Естественно, что подобный подход не отражает всей структуры класса, однако он позволяет судить о том, алгоритмы какого класса дают больший показатель качества в сравнении с другими классами (предполагается, что внутри одного класса показатели качества различаются слабее, нежели показатели между классами: $g(Alg_1, Alg_1) < g(Alg_1, Alg_2)$). После определения указанных *представителей класса* alg_1 и alg_2 предполагается сравнение именно этих представителей.

Таким образом, для трассы $T \in \mathbb{T}$ можно определить *класс наилучшего алгоритма*, а затем выбрать **наиболее эффективный** алгоритм среди класса, то есть тот, среднее значение функции качества Q для трасс из $\mathbb{T}$ является наибольшим.

Введение понятия зазора является важным в контексте нахождения *разделимых* классов алгоритмов. Считается, что классы алгоритмов Alg_1 и Alg_2 **разделимы**, если зазор между этими классами не меньше некоторого порогового значения p, устанавливаемого эмпирически: $g(Alg_1, Alg_2) \geq p$. Класс алгоритмов Alg считается **однородным**, если $g(Alg, Alg) < p$.

Естественным критерием для выбора классов алгоритмов является их попарная *разделимость*. Кроме того, дополнительным критерием является требование, чтобы каждый из этих классов был *однороден*.

4.6 Проверка однородности и разделимости

Для проверки *однородности* классов необходимо зафиксировать множество трасс $\mathbb{T}$. Считается, что данное множество было выбрано, а именно зафиксированы трассы длиной $10^3, 10^4$ и 10^5 (по 10^3 трасс каждой длины).

Требуется также определить необходимые классы алгоритмов. Выделяются 5 основных классов: основанные на *LRU*, основанные на *LFU*, основанные на *CLOCK*, основанные на *FIFO*, а также *оптимальные* стратегии (или, иначе говоря, алгоритмы, имитирующие алгоритм *Belady*). Алгоритмы, не входящие ни в один из упомянутых классов, выделяются в отдельный класс. Предполагается, что он является неоднородным (для проверки подобного предположения достаточно найти два алгоритма, зазор между которыми будет не меньше значения p, определённого в ходе проверки однородности остальных классов).

Как было указано в начале обзора, промоделировать работу алгоритмов можно с помощью *симулятора*, результатом работы которого является файл (для каждого алгоритма и для каждой трассы – отдельные файлы), содержащий информацию об изменении показателя качества в ходе работы алгоритма, а именно в каждой строчке такого файла содержится количество обработанных запросов и показатель качества на момент обработки этого количества запросов.

Так, например, для подобного файла допустимы следующие строчки:

$$15000 : 0.57$$
$$16000 : 0.59$$

В ходе запуска симулятора на всех трассах из выбранного множества (для каждой из которых запускался как минимум один представитель из 5 выделенных классов):

1. Для каждой из трасс был определён алгоритм кэширования с наибольшим показателем качества (то есть была произведена *разметка* трасс);
2. Была определена величина параметра $p = 0.01$;
3. Были рассчитаны зазоры для представителей соответствующих классов;
4. Были определены классы алгоритмов, обладающие свойством однородности.

4.7 Результаты обзора

Согласно описанным выше требованиям, предъявляемым к алгоритма кэширования, было решено проверять те алгоритмы, принцип работы которых различается (а именно – ввести разделение на классы). Так, например, LRU-кэш и $SNLRU$ относятся к одному классу, так как второй алгоритм основывается на первом. Однако LFU и LRU принадлежат разным классам, поскольку первый основывается на частоте запросов к объекту, а второй – на времени запроса. Предлагается выбирать в каждом классе алгоритмов одного представителя.

Таким образом, отбор алгоритмов не является независимым: выбор представителя alg класса Alg приводит к тому, что остальных представителей класса Alg выбрать не получится, если следовать требованиям, данным выше. По этой причине предполагается выбор представителей класса, максимизирующих зазор между классами.

Одним из важных критериев для выбора классов алгоритмов является *однородность* и *попарная разделимость*, то есть для каждой пары классов должно выполняться описанное в предыдущих подразделах свойство *разделимости*. Кроме того, для алгоритма alg в качестве требования к свойству разделимости добавляется наличие иных алгоритмов кэширования, функция качества для которых на некотором количестве трасс (минимум 10%) лучше, чем у alg.

Однако подобных критериев недостаточно: например, классы Alg_1 и Alg_2, каждый из которых состоит из одного алгоритма alg_1 и alg_2, функция качества для которых постоянна и принимает значения 0.2 и 0.4 соответственно, удовлетворяют требованиям попарной разделимость ($g(alg_1, alg_2) = 0.2$) и однородности ($g(Alg_1, Alg_1) = g(Alg_2, Alg_2) = 0$). Однако известно, что показатель качества для alg_2 выше, чем для alg_1.

Отсюда возникает критерий, которым является достижение алгоритмом максимума функции качества для достаточного количества трасс (для более чем 10% трасс из имеющихся, но не более 80%): критерий *качественности*.

Кроме того, также среди критериев выделяется *реализуемость* – возможность на практике реализовать описанный алгоритм кэширования. Так, например, оптимальный алгоритм на практике нереализуем (возможны лишь некоторые его приближения).

Результаты обзора представлены в таблице ниже.

Класс алгоритмов	Принцип работы	Реализуемость	Качественность	Однородность	Разделимость с иными классами
LRU-based	Свежесть запроса	+	+	+	+
LFU-based	Частота запроса	+	+	+	+
CLOCK-based	Цикличное размещение и флаги	+	±	+	+
FIFO-based	Очередь	+	−	±	+
Optimal	Запрос объекта в дальнейшем	∓	+	+	−
Other	Различен	?	?	−	?

Таблица 3: Таблица классов алгоритмов, составленная по результатам обзора

На основании указанного обзора были выбраны два класса алгоритмов (алгоритмы, основанные на LRU, и алгоритмы, основанные на LFU) как классы,

подходящие под все введённые критерии. Для этих классов были выбраны представители, для которых достигался наибольший зазор: ими оказались LRU и LFU (простейшие представители классов).

Стоит отметить следующую особенность введённых критериев. Выделить три класса алгоритмов, для которых выполнялось бы свойство разделимости, становится труднее. Подобные попытки либо приведут к уменьшению значения константы p (что, на самом деле, не будет говорить о наличии разделимости), либо же не будут успешными.

Подобный факт иллюстрируется рисунком 2.

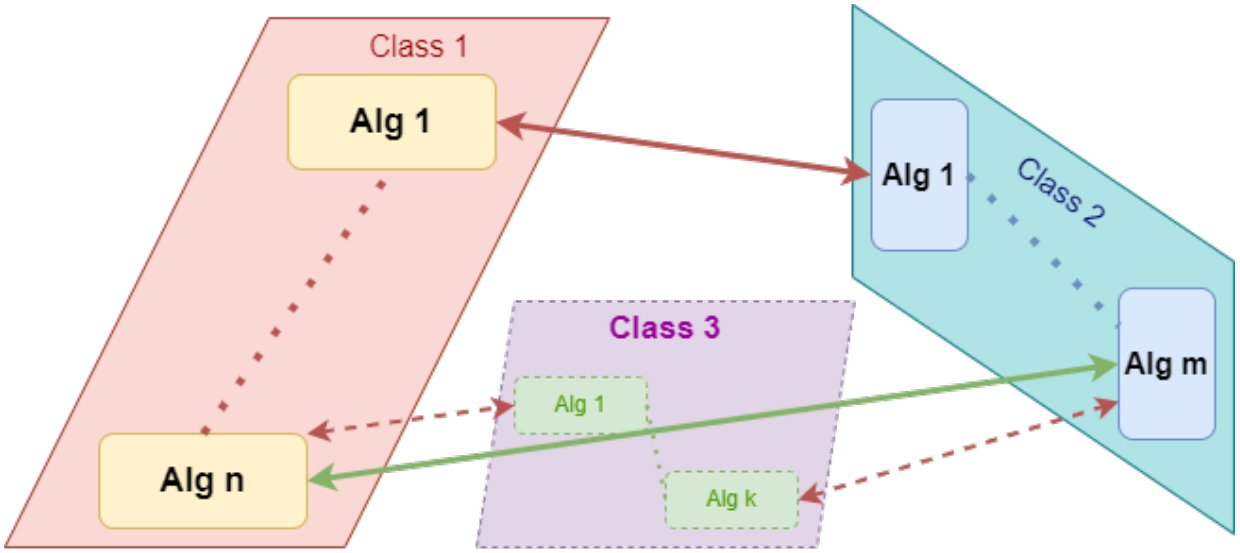

Рис. 2: *Разделимость для трёх классов алгоритмов*

Представляется возможным выделить среди двух классов таких представителей, чтобы их зазор был больше заданного p. Однако при попытке выделения таких представителей для трёх классов, зазор для каждого из классов уменьшается, что приведёт к потере свойства разделимости. Именно по этой причине было выбрано два класса алгоритмов кэширования.

Таким образом, основной задачей работы является построение и оценка качества классификатора набора данных (трасс) для нескольких алгоритмов кэширования (в частности, для двух алгоритмов).

5 Предлагаемый подход

5.1 Гипотеза локальности

С введённой функцией близости между элементами $\mathbb{T}$ выдвигается следующую гипотезу локальности (при фиксированных алгоритме alg, функции f, ρ_f и Q):

$$\rho_f(T_i, T_j) < \rho_f(T_i, T_k) \Rightarrow |Q(alg, T_i) - Q(alg, T_j)| < |Q(alg, T_i) - Q(alg, T_k)|,$$

$$\underset{alg}{\operatorname{argmax}} \Big(Q(alg, T_i)\Big) = \underset{alg}{\operatorname{argmax}} \Big(Q(alg, T_j)\Big) (1)$$

В общем случае такое следствие неверно. Однако предполагается, что 1 выполняется, то есть если трассы T_i и T_j ближе к друг другу в смысле ρ_f, чем T_i и T_k (иными словами, T_i более похожа на T_j, чем на T_k), то и разница между показателем качества при запуске на T_i и T_j и T_i и T_k одного и того же алгоритма будут различаться несильно и, кроме того, наибольшее значение функции качества для близких трасс достигается для одного и того же алгоритма.

5.2 Преобразование векторов

Выдвинутая гипотеза 1 может и не выполняться, если данные носят произвольный характер и имеют нерегулярную структуру. Это может быть обусловлено несколькими факторами.

1. Выбранная функция f неудачно отображает трассы в векторы: возможно, имеется большое количество неинформативных компонент, которые суммарно носят весомый вклад в сходство между трассами;

2. Функция сходства ρ_f не согласована с данными: либо при переводе в векторное пространство векторные представления объектов могут располагаться слишком далеко друг от друга (что приведёт к тому, что похожими друг на друга считаются, вообще говоря, достаточно разные в смысле их структуры трассы), либо же, наоборот, данные слишком близки друг к другу, то есть между отдельными группами схожих данных располагаются те, которые на них непохожи;

3. Функция измерения качества работы алгоритма Q неинформативна и не иллюстрирует результат работы алгоритма

Далее предполагается, что функция Q выбрана так, что она не вызывает указанных проблем. Считается, что основные затруднения могут возникать из-за функции f и, соответственно, ρ_f. В подобном случае осуществляется попытка "улучшить" структуру векторов v_f^T, расположив рядом с подобными друг другу векторами лишь те из них, что имеют общие черты, отделив тем самым группы похожих векторов.

Решать указанную задачу предлагается с помощью *однослойной нейронной сети*, основанной на функции *контрастной потери* (так называемой *Contrastive Loss*) [7].

Контрастная потеря определяется для пары объектов y_i и y_j, которым соответствуют некоторые метки классов l_i и l_j соответственно. Пусть $Y = \mathbb{I}_{\{l_i=l_j\}}$, $d = \rho(y_i, y_j)$, где ρ – введённая для таких объектов функция расстояния, $\mathbb{I}_{\{l_i=l_j\}} = \begin{cases} 1, & \text{если } l_i = l_j, \\ 0, & \text{иначе} \end{cases}$. Кроме того, пусть задано некоторое число $m > 0$ – "желаемое" расстояние между объектами с разными метками класса. Тогда

$$Contrastive\ Loss(y_i, y_j) = Yd^2 + (1 - Y)\max(0, m - d)^2$$

Иными словами, данная функция потерь учитывает, имеют ли объекты одинаковые метки классов: если да, то ненулевым будет первое слагаемое, где учтено расстояние между похожими объектами (между схожими элементами расстояние требуется уменьшить). В противном случае ненулевой будет вторая компонента, чтобы отделить непохожие объекты друг от друга: чем меньше это расстояние, тем больше значение функции потерь.

Для выборки $\mathbb{Y}$ функция потерь определяется через сумму потерь всех возможных пар:

$$Contrastive\ Loss(\mathbb{Y}) = \sum_{y_i \in \mathbb{Y}, y_j \in \mathbb{Y}} Contrastive\ Loss(y_i, y_j).$$

В теории именно минимизацией функции контрастной потери достигается желаемый результат по разделению данных.

Пусть имеется разделение выборки $\mathbb{Y}$ на обучающую $\mathbb{Y}_{train}$ и тестовую $\mathbb{Y}_{test}$, а также метки классов объектов из соответствующих выборок L_{train} и L_{test}. Пусть каждый вектор в указанных выборках имеет размерность M, а после преобразования будет иметь размерность m. Вводится отображение $\mathfrak{N}(\mathbb{Y}_{train}, L_{train}, \rho_f)$, принимающее на вход обучающую выборку с метками классов объектов из неё, а также метрику для объектов указанных выборок. Указанное отображение действует в $\mathbb{R}^{M \times m}$, то есть $\mathfrak{N}$ ставит в соответствие тройке параметров матрицу, которая будет называться *матрицей весов* W. Тогда $\mathbb{Y}_{new\_train} = \mathbb{Y}_{train}W$, $\mathbb{Y}_{new\_test} = \mathbb{Y}_{test}W$, причём отображение действует так, что $Contrastive\ Loss(\mathbb{Y}_{new\_train}) \to \min$. Понятия матрицы, составленной из вектор-строк, соответствующих объектам из выборки, и самой выборки, здесь отождествляются.

При введённых условиях решаются следующие проблемы:

1. Так как $m \ll M$, то стоит ожидать, что, если неинформативные компоненты и останутся, то их станет гораздо меньше;

2. Похожие друг на друга данные будут располагаться ближе друг к другу, чем к непохожим, ввиду минимизации контрастной потери.

Здесь неявно используется предположение, что в выборках $\mathbb{Y}_{train}$ и $\mathbb{Y}_{test}$ элементы имеют похожую структуру, поэтому, минимизируя контрастную потерю на $\mathbb{Y}_{new\_train}$, мы также минимизируем её и на $\mathbb{Y}_{new\_test}$.

Учитывая введённые функции, гипотезу локальности 1 переформулируется на случай преобразованных векторов. Пусть $\mathfrak{T} = f(\mathbb{T})W$, $\mathfrak{T} = \{\mathfrak{T}_1, ..., \mathfrak{T}_n, ...\}$ (считается, что из векторов v_f^T, полученных из трасс $T \in \mathbb{T}$ действием на них функцией f, была составлена матрица). Для трасс $\mathfrak{T}_1, \mathfrak{T}_2, ...$ выбирается та же самая метрика ρ_f.

$$\rho_f(\mathfrak{T}_i, \mathfrak{T}_j) < \rho_f(\mathfrak{T}_i, \mathfrak{T}_k) \Rightarrow |Q(alg, T_i) - Q(alg, T_j)| < |Q(alg, T_i) - Q(alg, T_k)|,$$

$$\underset{alg}{\operatorname{argmax}} \Big(Q(alg, T_i)\Big) = \underset{alg}{\operatorname{argmax}} \Big(Q(alg, T_j)\Big) \quad (2)$$

Ввиду минимизации контрастной потери предполагается, что данная гипотеза является более сильной.

Схематично подобный подход с преобразованием можно изобразить с помощью рисунка 3.

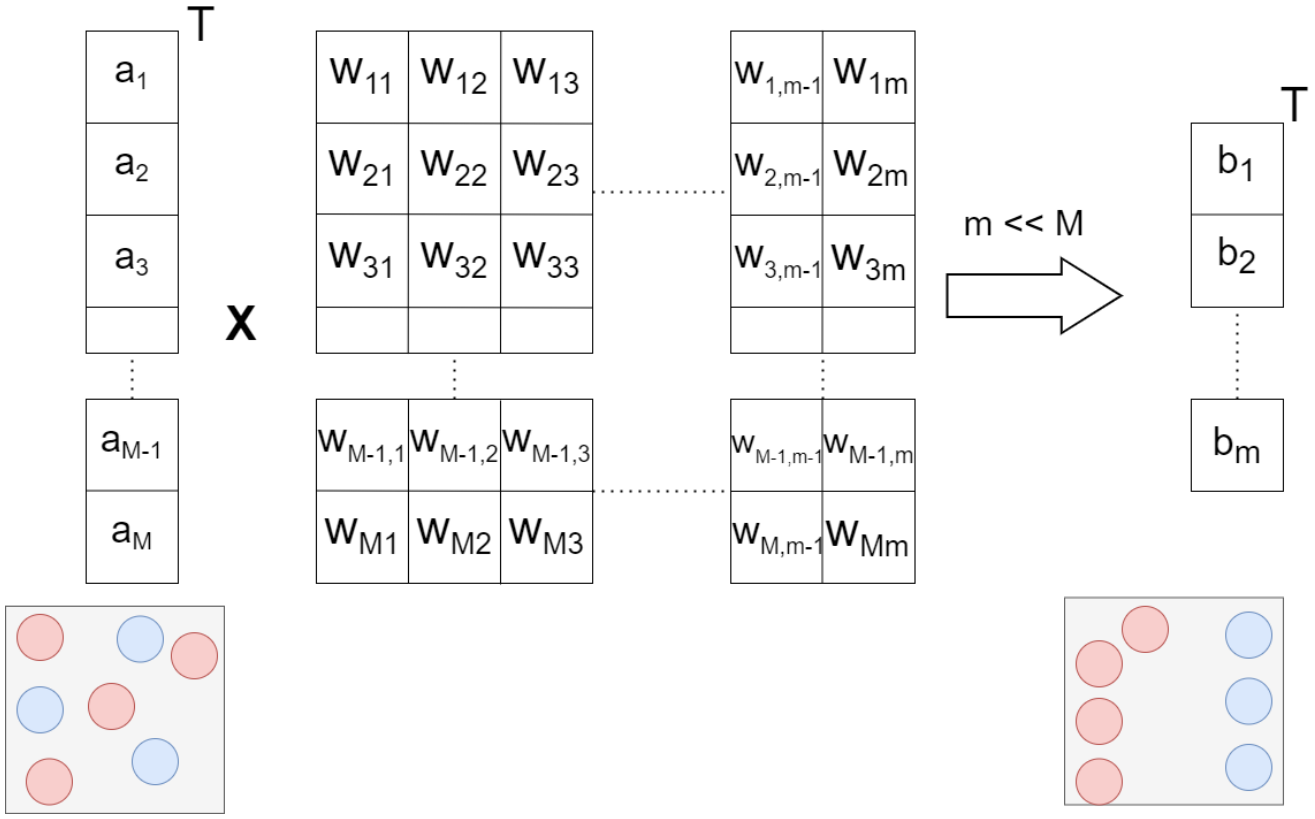

Рис. 3: *Подход для разделения данных с преобразованием векторов*

6 Экспериментальное исследование

6.1 План экспериментов

Далее конкретизируются функции, введённые в предыдущих разделах, для проведения экспериментов.

В качестве функции перевода трасс в векторы рассматриваются три отображения f_1, f_2, f_3. Чтобы преобразовать трассу, каждая из этих функций выделяет некоторые столбцы, после чего они преобразуются в вектор-строку. Так, f_1 выделяет столбец времени и размеров, f_2 - только размеров, а f_3 - только времени. Предполагается проверить, насколько успешным был выбор данных функций, а затем использовать лишь одну из них при дальнейшем исследовании. Считается, что эта функция обозначется как f.

Кроме того, также был выбран набор функций сходства для определения схожести полученных векторов. Основными фигурирующими в работе функциями сходства будут канберрское $\left(\rho(p,q) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|}\right)$, манхэттенское и евклидово расстояния.

Считается, что необходимое количество трасс (то есть 10^3) каждой длины (то есть $10^3, 10^4$ и 10^5) было выделено, при этом разделение на обучающую и тестовую выборку уже было произведено.

В работе рассматривается одна функция качества Q *BHR*, как и было оговорено ранее.

Учитывая результаты предыдущего раздела, выделяются два способа классификации исходных трасс:

1. Векторизация трасс функцией f и поиск ближайшего соседа из обучающей выборки;

2. Векторизация трасс, преобразование векторов для улучшения качества классификации и затем поиск ближайшего соседа из обучающей выборки.

Таким образом, существует два подхода к классификации, с помощью которых проверяются гипотезы локальности 1 и 2. Данная схема изображена на рисунке 4.

Сами эксперименты в ходе данной работы будут представлять собой обучение *классификатора*, основанного на методе k ближайших соседей, на *реальных данных* (трассах) с последующим поиском ближайшего соседа из обучающей выборки для объектов тестовой выборки и оценкой качества классификатора.

Проверка гипотезы локальности 1 осуществляется следующим образом:

1. Вычисление качества $Q(alg_j, T_i)$ на обучающей выборке: $\forall T_i \in \mathbb{T}^*_{train}$, $\forall alg_j \in Alg^*$;

2. Выбор другого, тестового, набора трасс $\mathbb{T}^*_{test} \subset \mathbb{T}^* : \mathbb{T}^*_{test} \cap \mathbb{T}^*_{train} = \emptyset$;

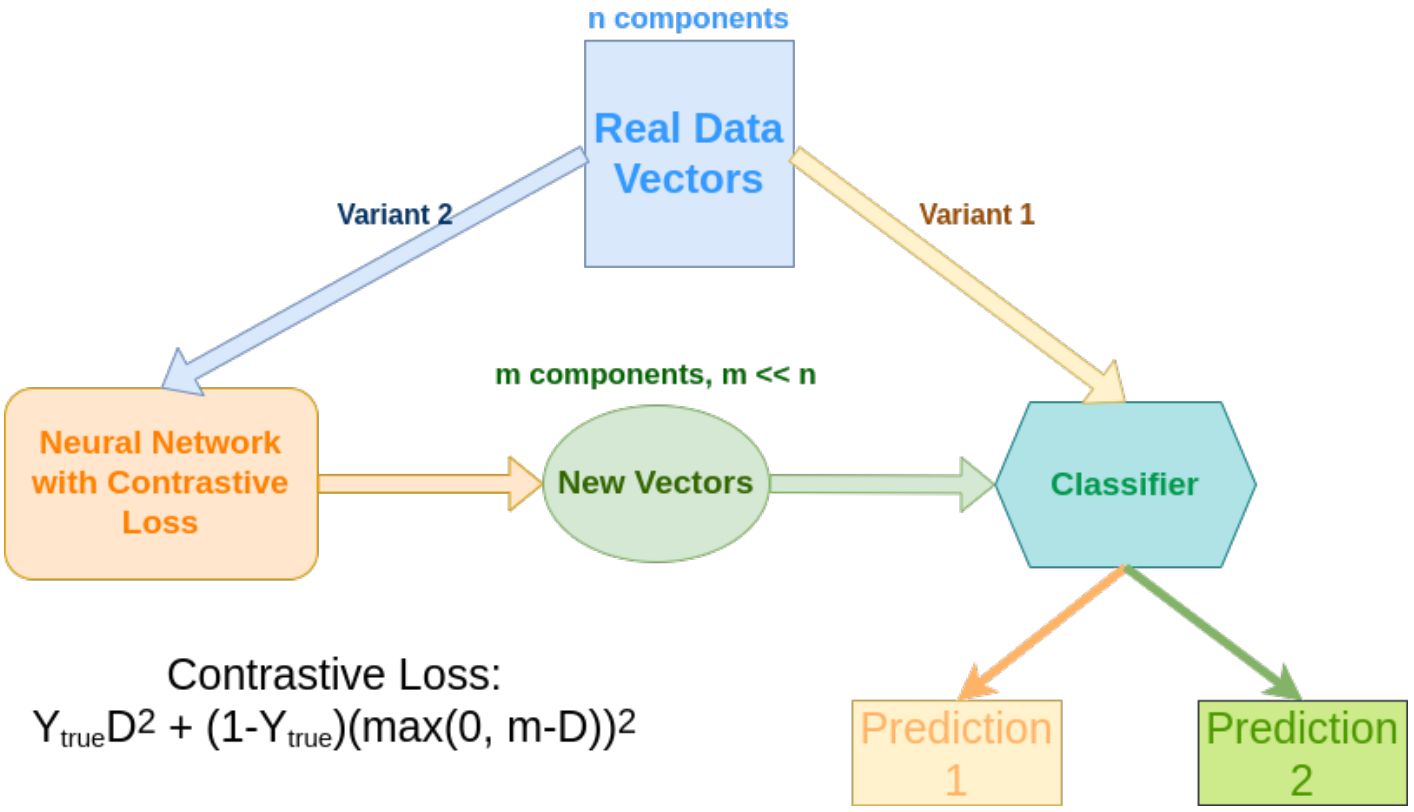

Рис. 4: *Общая схема подходов к классификации трасс*

3. Предсказание оптимального алгоритма кэширования $\forall T \in \mathbb{T}^*_{test}$ с предварительным обучением классификатора;

4. Непосредственное вычисление $Q(alg_j, T_i)$ $\forall T_i \in T^*_{test}$, $\forall alg_j \in Alg^*$;

5. Оценка качества полученного классификатора.

Аналогичным образом осуществляется проверка и гипотезы локальности 2. Проверка имеет похожий характер ввиду схожести самих гипотез. Для такой проверки необходимо наличие упомянутой ранее функции $\mathfrak{N}$. В качестве такой функции в программной реализации будет выступать однослойная нейросеть, принимающая на вход обучающую выборку и метки классов.

Данная сеть стремится к минимизации $ContrastiveLoss$ для всей выборки и возвращает веса, умножение которых на матрицу, составленную из векторов выборки, должно привести к улучшению качества классификации. Целевая размерность выбирается меньше, чем исходная: понижение размерности может как улучшить качество, так и уменьшить время, необходимое на предсказание.

Проверка гипотезы локальности 2 проводится похожим образом:

1. Вычисление качества $Q(alg_j, T_i)$ на обучающей выборке: $\forall T_i \in \mathbb{T}^*_{train}$, $\forall alg_j \in Alg^*$;

2. Обучение нейросети $\mathfrak{N}$ на обучающей выборке T^*_{train} и получение матрицы весов W;

3. Выбор тестового, набора трасс $\mathbb{T}^*_{test} \subset \mathbb{T}^* : \mathbb{T}^*_{test} \cap \mathbb{T}^*_{train} = \emptyset$ (нет пересечений с обучающей выборкой);

4. Изменение векторов из T^*_{test} путём умножения на матрицу W (для сравнения с обучающей выборкой необходимо привести векторы к одной размерности);

5. Предсказание оптимального алгоритма кэширования $\forall T \in \mathbb{T}^*_{test}$ с предварительным обучением классификатора;

6. Непосредственное вычисление $Q(alg_j, T_i)\ \forall T_i \in T^*_{test}, \forall alg_j \in Alg^*$;

7. Оценка качества полученного классификатора и сравнение с показателем качества без обучения.

6.2 Два алгоритма кэширования

Далее рассматривается простейший случай $Alg = \{LRU-based, LFU-based\}$. Пусть $alg_1 = LRU$, $alg_2 = LFU$.

6.2.1 Исходные представления

Для того, чтобы предсказать наилучший алгоритм кэширования на заданной трассе, необходимо отранжировать остальные трассы по степени близости. После этого требуется использование алгоритма k-NN.

k-NN (k-nearest neighbors). Воспользуемся алгоритмом k ближайших соседей [8]: классифируемый объект относится к тому классу, какой имеют большинство из k его ближайших "соседей" (иначе говоря, из k ближайших к нему элементов). Если их количество одинаково, то класс выбирается случайным образом. Если гипотеза локальности 1 верна, то тогда классификация с помощью данного алгоритма должна давать приемлемый результат, так как похожие трассы расположены достаточно близко.

В контексте определений, данных в предыдущем разделе, считается, что рассматриваются множества r-ых ближайших соседей так, чтобы $|N_r| \leq k$. По меткам классов, соответствующим трассам из построенного множества, делается вывод о том, к какому классу должна относиться трасса, не имеющая метки.

Для проверки гипотезы локальности 1 были рассмотрены различные функции сходства для векторов, полученных с помощью упомянутых выше функций f_1, f_2, f_3. Однако точность результатов предсказания иногда не превышала 0.5, как будет видно из таблицы результатов в дальнейшем. Это может говорить о том, что гипотеза локальности 1 неверна. По этой причине было решено перейти к использованию нейросети и изменению векторов.

6.2.2 Изменение векторных представлений

Для изменения размерности необходимо также выбрать целевую размерность. В экспериментах рассматривались, как правило четыре размерности: $5, 10, 20$ и 100.

Для того, чтобы выяснить, возможно ли действительно говорить о локальности, одним из подходов является применение визуализации с предварительным понижением размерности векторных представлений объектов. Однако здесь стоит отметить важный момент: поскольку нейросеть минимизирует констрастную потерю, то функция сходства, которую использует нейросеть, и функция сходства, используемая для визуализации, должны быть согласованы.

При визуализации же несогласованных функций сходства полученные картины зачастую проблематично интерпретировать. Причиной этого могут быть потеря качества при сильном понижении размерности (понижение размерности до

двух может привести к потере большого количества информации), несогласованность функций сходства или же не лучшая структура данных.

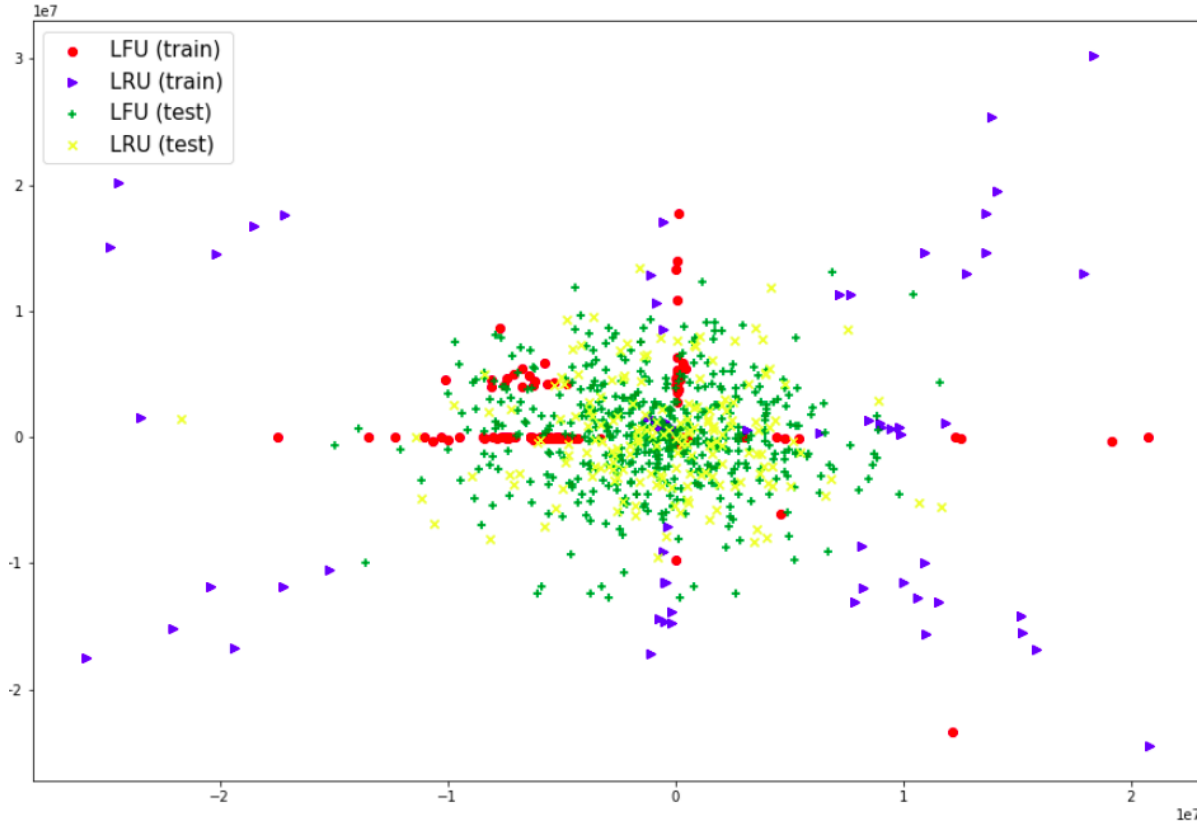

Рис. 5: *Визуализация при обучении с канберрским расстоянием*

Так, например, по рисунку 5 делается вывод, что при визуализации локальности вовсе не наблюдается. Предполагается, что такой результат возникает из-за слишком сильного понижения размерности.

6.3 Результаты экспериментов для двух алгоритмов

Общие результаты экспериментов представлены в таблице 4. На указанной таблице идёт последовательное "ветвление": сначала по длине потока запросов, затем по функции сходства, после чего – по размеру кэша и, наконец, непосредственно по целевой размерности при её понижении. Результаты, которые после обучения становятся лучше, чем до него, обозначены зелёным; те, что ухудшаются, – красным. Если изменения незначительные, то выбрано жёлтое обозначение. Аналогичные обозначения приняты для исходного качества (если высокое – то зелёное и так далее). Результаты представлены для функции f_2, выделяющей столбец размеров объектов в потоке запросов, и для случая $k = 1$ (то есть классификация производится по ближайшему соседу).

Анализ указанной таблицы позволяет судить о структуре данных и о необходимости применения обучения к ним. Из анализа таблицы делается вывод, что далеко не всегда обучение нейросети даёт выигрыш в контексте качества класси-

Длина потока запросов	Метрика	Размер кэша	Исходное качество	Целевая размерность	Качество после обучения
10000	Canberra	64 Gb	0.69	5	0.71
				10	0.728
				20	0.716
				100	0.724
		256 Gb	0.318	5	0.736
				10	0.74
				20	0.741
				100	0.714
		512 Gb	0.384	5	0.57
				10	0.687
				20	0.718
				100	0.71
	Manhattan	64 Gb	0.722	5	0.54
				10	0.512
				20	0.484
		256 Gb	0.478	5	0.567
				10	0.481
				20	0.534
1000	Canberra	10 Gb	0.719	5	0.473
				10	0.356
				20	0.4
		32 Gb	0.547	5	0.49
				10	0.507
				20	0.512
	Euclid	10 Gb	0.744	5	0.5
				10	0.541
				20	0.553
		32 Gb	0.75	5	0.573
				10	0.57
				20	0.583
100000	Canberra	64 Gb	0.86	5	0.869
				10	0.866
				20	0.871
		128 Gb	0.604	5	0.714
				10	0.716
				20	0.715
	Manhattan	64 Gb	0.749	5	0.724
				10	0.69
				20	0.662
		128 Gb	0.727	5	0.6
				10	0.52
				20	0.515

Таблица 4: *Таблица с результатами экспериментов для двух алгоритмов кэширования*

фикации. Так, например, на малой длине потока запросов (1000 запросов) выигрыша по качеству вовсе нет никакого (наблюдается лишь ухудшение качества).

Отсюда следует вывод вывод, что, если качество классификации изначально низкое, то обучение только улучшит его: тогда применение нейросети имеет смысл почти всегда. Если же качество достаточно высоко (≥ 0.7), то тогда обучение, как правило, его не улучшает, а в худшем случае ещё и понижает. Если же качество среднее (порядка 0.5), то тогда качество, если и меняется, то несильно (но, как правило, в сторону увеличения).

Следовательно, на основе этих данных можно предположить, что справедливым будет следующий принцип:

- Если показатель качества классификации ≥ 0.7, то тогда обучение, как правило, не приводит к увеличению качества;
- Если показатель качества классификации достаточно низок (≤ 0.3), то обучение почти всегда приводит к увеличению качества;
- При результате, близком к ≥ 0.5, применение обучения несущественно, к изменению качества, как правило, не приводит.

На основе вышеописанного предлагается следующий подход:

1. Для выбора наилучшего алгоритма кэширования используется классификация;
2. Объекты преобразуются в векторы;
3. Для полученных векторов ищется ближайший сосед с уже известной меткой класса;
4. Опционально: обучение нейросети, основанной на функции контрастной потери, по полученным векторам для улучшения точности классификации.

Таким образом, для выбора наилучшего алгоритма кэширования достаточно сначала определить класс этого алгоритма (путём отбора отдельных представителей классов), а затем выбрать алгоритм с наибольшим средним показателем качества для обучающей выборки. Изначальныый выбор алгоритмов с наибольшим средним BHR может привести к тому, что классификация будет иметь низкую точность из-за слишком малого зазора между такими представителями классов.

6.4 Три алгоритма кэширования

Как было упомянуто ранее, отбор трёх классов алгоритмов может привести к тому, что зазор между классами будет уменьшаться, что может негативно сказываться на точности классификации. Тем не менее, для проведения подобных экспериментов было выделено три класса алгоритмов (помимо $LRU - based$ и $LFU - based$ был также выделен класс $CLOCK - based$) и выбраны представители классов (в классе $CLOCK - based$ им стал алгоритм $CLOCK$).

Изменение векторных представлений используется в точности же такое, что и для двух алгоритмов кэширования.

6.5 Результаты экспериментов для трёх алгоритмов

Как и для двух алгоритмов кэширования, результаты представлены в итоговой таблице 5 (её структура полностью аналогична таблице для двух алгоритмов).

Как и в случае двух алгоритмов, в таблице изображены результаты для функции f_2, выделяющей в таблице столбец размеров. Классификация производилась по методу ближайшего соседа (то есть параметр $k = 1$).

Стоит отметить, что в случае трёх алгоритмов кэширования обучение нейросети, если качество классификации достаточно высоко (порядка 0.7) иногда его улучшает (как, например, в случае длины потока запросов в 10^4 и при размере кэша в 10 Gb), несмотря на уменьшение зазоров классов. Для малой же длины потока запросов и малого размера кэша (10^3 и 10 Gb соответственно) качество падает (изначально оно было высоким, близким к 0.7), а для средней длины потока запросов и среднем размере (10^4 и 128 Gb) в целом не изменяется (что, впрочем, согласовывается с принципом изменения качества классификации для двух алгоритмов кэширования). Следовательно, для канберрского расстояния при достаточно большом размере потока запросов (например, при 10^4) или при достаточном большом размере кэша (например, при 128 Gb) имеет смысл обучать нейросеть, даже если изначальное качество было высоко (близко к 0.7).

Однако для евклидовой и манхэттенской метрики результаты, в сравнении со случаем двух алгоритмов, как правило, ухудшаются. Это может говорить о том, что данные функции близости не предназначены для классов алгоритмов с небольшим зазором. Канберрское же расстояние, напротив, может разделять классы даже с небольшим зазором.

Тем не менее, в целом принцип изменения качества для канберрского расстояния сохраняется (с небольшой поправкой для высокого качества классификации):

- Если показатель качества классификации низкий (≤ 0.3), то обучение приводит к увеличению качества;
- При результате, близком к 0.5, применение обучения не даёт существенного увеличения или уменьшения качества классификации (как правило, качество не изменяется);

- Если показатель качества классификации близок к 0.7, то тогда обучение может как увеличить, так и уменьшить качество: при малой длине потока запросов с малым размером кэша, как правило, будет наблюдаться ухудшение качества. В остальных случаях наблюдается рост (возможен как незначительный рост (порядка 0.05), так и достаточно весомый – более 0.1).

Таким образом, для классификации в случае трёх алгоритмов кэширования лучше всего использовать канберрское расстояние. Если изначально размера кэша и длина потока запросов малы, то тогда обучение однослойной нейросети, скорее всего, не приведёт к улучшению качества. В остальных же случаях качество в целом улучшается (хоть, возможно, и незначительно), поэтому обучение нейросети имеет смысл.

Длина потока запросов	Метрика	Размер кэша	Исходное качество	Целевая размерность	Качество после обучения
1000	Canberra	10 Gb	0.7057	5	0.384
				10	0.424
				20	0.496
		128 Gb	0.36	5	0.525
				10	0.485
				20	0.481
	Euclid	10 Gb	0.656	5	0.264
				10	0.289
				20	0.296
		128 Gb	0.357	5	0.36
				10	0.335
				20	0.281
10000	Canberra	10 Gb	0.669	5	0.8
				10	0.803
				20	0.777
				100	0.645
		128 Gb	0.404	5	0.39
				10	0.387
				20	0.426
				100	0.439
		256 Gb	0.294	5	0.521
				10	0.506
				20	0.476
				100	0.504
	Manhattan	10 Gb	0.374	5	0.441
				10	0.411
				20	0.457
		256 Gb	0.429	5	0.374
				10	0.366
				20	0.371
100000	Canberra	64 Gb	0.794	5	0.863
				10	0.864
				20	0.861
		128 Gb	0.317	5	0.704
				10	0.705
				20	0.693
	Manhattan	64 Gb	0.729	5	0.507
				10	0.49
				20	0.583
		128 Gb	0.64	5	0.409
				10	0.361
				20	0.36

Таблица 5: *Таблица с результатами экспериментов для трёх алгоритмов кэширования*

7 Результаты работы

В ходе работы были получены следующие результаты:

1. Проведён обзор алгоритмов кэширования, по результатам которого определены два класса алгоритмов (*LRU-based* и *LFU-based*), выбраны представители указанных классов (стратегии LRU и LFU) для изучения показателя качества алгоритмов на реальных данных с целью выбора наилучшего алгоритма кэширования. Также произведена разметка данных путём запуска симулятора алгоритмов кэширования на заданном наборе трасс;

2. На основе проведённого обзора алгоритмов кэширования были выделены три класса алгоритмов *LRU-based*, *LFU-based* и *CLOCK-based*), выбраны представители указанных классов (стратегии LRU, LFU и $CLOCK$) и, как и для случая двух алгоритмов, произведена разметка данных;

3. Сформулирована гипотеза локальности, заключающуюся в предположении близости трасс похожей структуры, на основе которой строится подход к выбору наилучшего алгоритма кэширования;

4. Предложен подход к выбору наилучшего алгоритма кэширования, основанный на классификации, которая, в свою очередь, основывается на векторизации и изменении исходных векторных представлений;

5. Проведено экспериментальное исследование, в том числе с использованием однослойной нейронной сети, подтверждающее корректность предложенного подхода к выбору наилучшего алгоритма кэширования.

8 Заключение

Предложенный подход к выбору наилучшего, то есть имеющего наибольший показатель качества, алгоритма кэширования может быть полезен для экономии времени при выборе наилучшего из алгоритмов для повышения показателя качества (BHR). Тем самым, подход к выбору наилучшего алгоритма заключается в формировании обучающей выборки, последующей классификации поступающих данных и выборе в среднем наилучшего алгоритма по запускам и прогнозам.

Стоит отметить, что в работе применяется общий подход к алгоритмам, данным, функциям качества и функциям сходства. Следовательно, представляется возможным обобщить используемый подход для иных классов алгоритмов (необязательно алгоритмов кэширования), функций качества и входных данных.

Возможным развитием работы является исследование подхода, связанного с синтезом алгоритма, позволяющего получить максимальный показатель качества для заданных данных.

Список литературы

1. Zhou Y., Philbin J., Li K. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches //USENIX Annual Technical Conference, General Track. – 2001. – С. 91-104.

2. Megiddo N., Modha D. S. ARC: A Self-Tuning, Low Overhead Replacement Cache //Fast. – 2003. – Т. 3. – №. 2003. – С. 115-130.

3. Bansal S., Modha D. S. CAR: Clock with Adaptive Replacement //FAST. – 2004. – Т. 4. – С. 187-200.

4. Атрибутивно-сегментный кэш. Git-репозиторий [Электронный ресурс]. — URL: https://github.com/himotokun/scache (дата обращения: 05.10.2020).

5. Belady L. A. A study of replacement algorithms for a virtual-storage computer //IBM Systems journal. – 1966. – Т. 5. – №. 2. – С. 78-101.

6. Jain A., Lin C. Back to the future: Leveraging Belady's algorithm for improved cache replacement //2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). – IEEE, 2016. – С. 78-89.

7. Wang F., Liu H. Understanding the behaviour of contrastive loss //Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. – 2021. – С. 2495-2504.

8. Guo G. et al. KNN model-based approach in classification //OTM Confederated International Conferences"On the Move to Meaningful Internet Systems". – Springer, Berlin, Heidelberg, 2003. – С. 986-996.

A Приложение

A.1 Исходные коды используемых программ

Исходные коды приложений доступны в репозитории GitHub: `https://github.com/Tyapkins/RealData`