Московский государственный университет
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Важдаев Александр Сергеевич

Стратегия кэширования, использующая разделение кэша на основании атрибутов информационных объектов

Выпускная квалификационная работа

Научный руководитель:
математик ЛВК
А. М. Колосов

Москва, 2020

Аннотация

В работе выдвигается гипотеза о сравнительной эффективности алгоритма кэширования, основывающегося на анализе семантических метаданных (атрибутов) хранимых информационных объектов и временном распределении запросов к кэшу, строится такой алгоритм и проводится серия экспериментов с целью проверки этой гипотезы.

Содержание

1 Введение и обзор

Кэширование используется в самых различных системах для оптимизации работы тех или иных частей — начиная с кэш-памяти процессоров и до отдельных гигантских кэш-серверов в сетях типа CDN.

Как правило, задача кэша заключается в том, чтобы максимально сократить долю *промахов* — обращений к такому объекту, который не содержится в кэше. Для этого нужно наполнять кэш-память подходящими объектами. Подходов к выбору таких объектов существует достаточно много, причём разные подходы более эффективны для тех или иных областей применения, чем другие.

Принцип работы конкретной кэширующей системы чаще всего задаётся некоторым алгоритмом, который называется ***стратегией кэширования***.

1.1 Анализ известных стратегий

Простейшие стратегии Самые простые в реализации, однако, не всегда достаточно эффективные для применения в реальных системах — такие, как, например, LRU (Least Recently Used) и LFU (Least Frequently Used). Обычно они основываются на простейшем подсчёте популярности или полезности объекта: в LRU из кэша удаляется объект, который не запрашивался дольше всех остальных, в LFU — объект, который запрашивался реже остальных.

Эти стратегии часто являются основой для построения других, более сложных стратегий.

Сегментные кэши Как правило, гораздо более эффективные стратегии, но по-прежнему достаточно простые в реализации. Самый популярный пример — SNLRU (N-Segmented LRU). Главная особенность таких стратегий заключается в *сегментировании* — разделении кэша на изолированные части.

В случае SNLRU кэш делится на N сегментов, между которыми определённым образом перемещаются объекты, причём внутри каждого из сегментов используется стратегия LRU. В этой стратегии сегменты делятся по *приоритетности* — самым приоритетным считается «верхний» сегмент, в противоположность «нижнему», и более приоритетные запросы перемещаются между сегментами «снизу вверх» (и, соответственно, наоборот), а из самого «нижнего» сегмента объекты удаляются окончательно. Подобным образом можно организовать сегментирование практически любой стратегии, но наиболее распространена именно схема с LRU.

TrendLearner Одна из нестандартных стратегий, описанная в [4]. В отличие от остальных, **не является стратегией общего типа**, то есть подходит лишь для объектов определённого класса, а не любых вообще.

Основа механизма TrendLearner базируется на *атрибутах объектов* — некоторой семантической метаинформации, приписанной к объектам. В данном случае описывается работа механизма для видеохостинга, соответственно, речь идёт об атрибутах типа «автор видео/режиссёр фильма», «дата премьеры» и т. п.

Система кэширования анализирует группы объектов с одинаковыми атрибутами и производит их кластеризацию, после чего каждому кластеру сопоставляется его приоритетность, и в кэш сохраняются по возможности объекты с наибольшим приоритетом.

Главный недостаток конкретно этой системы — неприспособленность к изменениям и не вполне корректная обучаемость. Приоритеты кластеров, вообще говоря, могут довольно быстро и значительно изменяться, и система обучения не успевает приспособиться к изменениям.

AdaptSize Достаточно простая, но всё же нестандартная стратегия, которая описывается в [5]. Довольно очевидная стратегия, основанная на фильтрации объектов по пороговому значению определённого признака — в данном случае, по размеру объекта. Суть алгоритма: в процессе анализа запросов вычисляется *оптимальный пороговый размер*, после чего в кэш сохраняются только объекты, не превышающие этого размера.

Эффективность кэша основывается на гипотезе, что маленькие объекты более требовательны к скорости загрузки: пользователь может подождать несколько дополнительных секунд, ожидая загрузки двухчасового фильма, но при этом не готов долго ждать начала воспроизведения 30-секундного ролика.

Недостаток такого подхода в том, что он, вообще говоря, легко может оказаться неверным — какой-либо длинный видеофайл может оказаться настолько популярнее такого же по размерам множества коротких, что сохранить его, несмотря на размер, оказалось бы значительно эффективнее.

Абстрактные стратегии Для анализа эффективности кэша часто бывает полезно сравнивать разные стратегии не только друг с другом, но и с некоторым абсолютным (или почти абсолютным) эталоном. Для этого используются так называемые *оптимальные стратегии* или стратегии *с предвидением*. Особенность таких стратегий в том, что им известны не только запросы, которые уже произошли, но и те, которые произойдут в будущем. Такие стратегии являются чисто модельной абстракцией, поскольку, по понятным причинам, не могут существовать в реальности. В качестве примера можно привести алгоритм Белади [2], работающий по следующему правилу: *из кэша при необходимости удаляется объект, который дольше всего не будет запрошен в будущем.*

Разные стратегии кэширования принято сравнивать друг с другом по какому-либо показателю, отражающему качество работы кэша. Часто в роли такого показателя используется величина **hit rate** — соотношение *попаданий в кэш* ко всем

запросам в той или иной форме. Ввиду особенностей исходных данных в работе будет использоваться его разновидность **byte hit rate (BHR)**, что будет подробно описано далее [3].

Для тестирования кэш-системы, выявления качественных характеристик, иных показателей применяют *симуляцию*: запускается специальная программа-*симулятор*, которая получает на вход *историю запросов*, которые затем осуществляются к кэш-системе как если бы это были настоящие запросы от клиентов.

Предлагаемая в работе система комбинирует несколько вышеописанных подходов и основывается на идее *семантического разделения данных*, то есть, выбор предпочтений кэша будет основываться на смысловой нагрузке тех или иных объектов. Это могут быть, к примеру, метаданные сохраняемых файлов, какие-либо особенности их внутреннего строения и так далее. В дополнение к этому, система использует *сегментный подход* и *динамическое разделение во времени*. Выдвигается гипотеза, что в определённых ситуациях это может дать улучшение какого-либо качественного показателя системы кэширования.

Основной задачей работы будет построение такого алгоритма и выяснение, существуют ли такие входные данные, на которых этот алгоритм даст увеличение качественной характеристики (величины BHR).

2 Описание задачи и начальная гипотеза

2.1 Исходные данные

Для исследования были предоставлены несколько наборов реальных данных, представляющих собой историю запросов к нескольким узлам CDN некоторого видеохостинга. Объекты размечены ***атрибутами*** — некоторыми семантическими данными (такими как, например, жанр видео, режиссёр фильма и т. п.). Суммарный объём всех данных составляет порядка 10 Гб и включает более 537 тысяч размеченных объектов и историю около 75 миллионов запросов.

2.2 Мотивы

Если рассмотреть множество распределений во времени объёма запросов к объектам, соответствующих определённому атрибуту или множеству атрибутов, в некоторых из таких распределений можно обнаружить некое подобие *периодической зависимости*. Формально её можно описать как совпадение *с некоторой точностью* количества запросов (либо процентной доли или иной количественной характеристики) в промежутки времени, отличающиеся на некоторый фиксированный промежуток. Такую периодическую зависимость далее будем называть ***мотивом***, а атрибуты объектов, образующих такую зависимость — ***атрибутами этого мотива***.

2.3 Особенности системы кэширования

Можно выделить три основных отличительных черты предлагаемой системы:

- **Сегментирование.** Всё пространство кэша делится на сегменты определённым образом. Внутри каждого сегмента используется какая-либо стратегия кэширования так, будто каждый сегмент является самостоятельным независимым кэшем; при этом каждому сегменту строго соответствует какое-либо подмножество всех возможных объектов;

- **Атрибуты.** Каждому сегменту кэша сопоставлен некоторый набор атрибутов; тогда соответствие объекта тому или иному сегменту определяется по атрибутам этого объекта;

- **Мотивы.** Распределение сегментов строится динамически на основе *мотивов*, которые отыскиваются алгоритмом среди поступающих запросов: обнаруженные циклические зависимости используются для предсказания возможного распределения запросов по атрибутам в предстоящий момент времени.

2.4 Эксперименты, генератор данных

Поскольку такая система кэширования, вообще говоря, пригодна к использованию не только для конкретной ситуации, а вообще для любых размеченных данных, стоит проверить её работу не только на нескольких конкретных наборах данных, а и на множестве каких-либо других.

Так как реальных наборов предоставлено всего несколько штук, можно синтезировать новые наборы данных на основе существующих. Поскольку точно известны закономерности в данных, определяющие целесообразность применения новой системы, можно реализовать ***генератор наборов данных***, обладающих такими закономерностями в той или иной степени. Характер этих закономерностей при этом будет определяться ***параметрами этого генератора***.

С помощью такого генератора можно создать целое множество наборов данных и определить, при каких параметрах достигается улучшение качественных показателей разработанного алгоритма кэширования (и достигается ли вообще).

2.5 План работы

В общих чертах, план предстоящей работы следующий:

1. **Построить систему кэширования**, использующую метод сегментирования на основании мотивов — периодических закономерностей в количестве запросов к объектам, имеющим заданный набор атрибутов;

2. **Построить экспериментальную среду**: симулятор запросов к кэшу по заданному набору запросов и генератор таких наборов по некоторым численным параметрам;

3. **Провести тестирование** кэша на построенных наборах данных, сравнить качественные характеристики получившейся системы с некоторой базовой системой;

4. **Сделать вывод** о применимости гипотезы — действительно ли такой подход может повышать качество кэширования, и, если да, то при каких условиях (характеристиках алгоритма, параметрах генератора) это достигается.

3 Постановка задачи, определения

Для формализации вышеописанной системы и постановки задачи введём базовые определения, на которых будет основываться работа.

3.1 Объекты и атрибуты

Главным понятием всей работы является ***информационный объект***. Будем считать, что фактическая информационная нагрузка объекта не важна; достаточно, чтобы объект имел некоторый ***идентификатор*** — элемент произвольного множества идентификаторов (в работе в качестве идентификаторов используются текстовые строки) такой, чтобы каждый идентификатор был уникальным для данного объекта, т. е., для каждого элемента из множества идентификаторов существовал ровно один объект с таким идентификатором, а также у каждого объекта должен быть задан ***размер*** — некоторое положительное число.

Также потребуется понятие ***атрибута*** и ***пространства атрибутов***. ***Пространство атрибутов*** подобно множеству идентификаторов: это пространство произвольных элементов, для которых важно лишь то, принадлежит элемент этого пространства некоторому множеству или нет. Подобно идентификаторам, в работе в качестве пространства атрибутов используется заданное множество текстовых строк.

Для каждого объекта задаётся ***множество атрибутов объекта*** — конечное число элементов из пространства атрибутов. Говорят, что объект обладает данным атрибутом, если его множество атрибутов содержит этот атрибут.

Таким образом, информационный объект можно задать как тройку $\omega = \{I,\ W,\ \Xi\}$, где I_ω — идентификатор объекта, $W_\omega > 0,\ W_\omega \in \mathbb{R}$ — размер объекта, $\Xi_\omega = \{\xi_\omega\}$ — атрибуты объекта.

Множество всех заданных объектов обозначим как $\Omega = \{\omega\}$, причём верно, что $\forall \omega_1, \omega_2 \in \Omega\ \omega_1 \neq \omega_2 \Leftrightarrow I_{\omega_1} \neq I_{\omega_2}$, а пространство атрибутов как $\Xi = \{\xi\}$.

Множество всех атрибутов с сопоставленными каждому атрибуту объектами будем называть ***таблицей атрибутов***: $\tau :\ \Xi \to \{\omega_\Xi\}$.

3.2 Кэш и события

Ещё одним центральным определением является ***кэш***. Кэш понимается как некоторый объект, для которого задано ***состояние кэша*** (далее эти понятия используются как тождественные).

Кэш содержит в себе некоторые объекты, то есть, для кэша необходимо задать множество объектов, которые в нём содержатся на данный момент.

Поскольку дисковое пространство кэша конечно, для кэша также зададим ***размер кэша*** — положительное число, такое, что суммарный размер всех содержащихся в кэше объектов не превосходит этого размера.

Таким образом, кэш задаётся как пара $\mathcal{C} = (\Psi,\ M)$, где $\Psi \subseteq \Omega$ — объекты кэша, $M > 0,\ M \in \mathbb{R}$ — размер кэша, причём $\sum_{\psi \in \Psi} W_\psi \leqslant M$.

Подчёркиваем, что кэш — не более чем множество содержащихся объектов и заданный максимальный размер, для него не заданы какие-либо операции или правила взаимодействия с иными сущностями.

Независимо от определения кэша вводится понятие ***события***. Итак, ***событие*** — это пара $\varepsilon = (\square,\ t)$, где $t \in T$ — ***момент времени*** из некоторого ***временного пространства***; в качестве временного пространства может быть задано любое множество, для элементов которого определены операции сравнения (т. е., для $\forall t_1, t_2 \in T$ либо $t_1 > t_2$, либо $t_1 < t_2$, либо $t_1 = t_2$); это может быть, например, множество натуральных или вещественных чисел, или некоторое его подмножество; в работе пространство времени задано как множество целых чисел в диапазоне $\left[0, 2^{32} - 1\right]$. $\square \in \{Q\left\langle\omega\right\rangle, S\left\langle M\right\rangle\}$ называется ***операцией события***, где $Q\left\langle\omega\right\rangle$ — ***запрос объекта*** ω, а $S\left\langle M\right\rangle$ — ***установка максимального размера*** M.

Уточнение: сами по себе события не несут какой-либо смысловой нагрузки, это лишь абстрактные обозначения. События не имеют никакой связи с кэшем или его частями.

Определим ***историю событий*** как произвольное множество событий $E = \{\varepsilon\}$, а также:

- ***История событий*** E ***до момента*** t *(включительно)*: $E_t \subseteq E : \varepsilon \in E_t \Leftrightarrow t_\varepsilon \leqslant t$.
- ***История событий*** E ***до события*** ε: $E_{\underline{\varepsilon}} \subseteq E : \forall \tilde{\varepsilon} \in E\ \tilde{\varepsilon} \in E_{\underline{\varepsilon}} \Leftrightarrow t_{\tilde{\varepsilon}} < t_\varepsilon$.
- ***История событий*** E ***после события*** ε: $E_{\bar{\varepsilon}} = E_{\underline{\varepsilon}} \cup \varepsilon$.

3.3 Функции кэширования и оценка кэша

Поскольку смысл *системы кэширования* заключается в том, чтобы, среди прочего, сохранять объекты и обрабатывать запросы клиента, необходимо задать некоторые *правила* работы такой системы. В нашем случае предполагаемая работа системы может быть описана примерно следующим образом: на вход подаётся некоторая история событий, которые обрабатываются согласно определённому алгоритму, после чего на выходе получается *состояние кэша после обработки этих событий*.

В качестве такого алгоритма зададим ***функцию кэширования*** $\mathfrak{R} : (\mathcal{C}, E) \to \mathcal{C}'$. То есть, функция кэширования по заданному начальному состоянию кэша и истории событий возвращает новое *состояние кэша после данной истории событий*.

Определим ***состояние кэша после события*** ε: $\mathcal{C}_\varepsilon = \mathfrak{R}\left(\mathcal{C},\ E_{\bar{\varepsilon}}\right)$.

Здесь описанные выше *события* приобретают фактический смысл. Во-первых, будем считать, что для кэш-функции выполняется ***корректность задания размера***: для события $\varepsilon = \left(S\left\langle M\right\rangle,\ t\right) \in E$ верно, что $M_{\mathcal{C}_\varepsilon} = M$ и $\mathcal{C}_\varepsilon$ удовлетворяет определению кэша (т. е., $\sum_{\psi \in \Psi_{\mathcal{C}_\varepsilon}} W_\psi \leqslant \mathcal{C}_\varepsilon$).

Во-вторых, для событий типа «запрос объекта» введём ещё одно понятие — ***результат запроса***. Пусть $\mathcal{C}'$ — состояние кэша в некоторый момент времени, $\varepsilon = \left(Q\left\langle\omega\right\rangle, t\right)$ — запрос. Тогда ***результат запроса*** ε ***к кэшу*** $\mathcal{C}'$ равен $\delta_{\mathcal{C}',\varepsilon} = \left[\omega \in \Psi_{\mathcal{C}'}\right],\ \delta_{\mathcal{C}',\varepsilon} \in \{\mathbf{true},\ \mathbf{false}\}$.

Рассмотрим частный случай функций кэширования, который потребуется для дальнейшей работы — ***сегментные функции кэширования***.

Итак, сегментная функция кэширования — это такая функция кэширования $\mathfrak{S} \in \{\mathfrak{R}\}$, что для $\mathfrak{S}$ определены ***множество сегментов*** $\Sigma_{\mathfrak{S}}(t) = \{\sigma_{\mathfrak{S}}\}$ и ***истории сегментов*** $\{E_\sigma(t)\}$, где σ — **кэш**, причём $\sum_{\sigma \in \Sigma_{\mathfrak{S}}} M_\sigma \leqslant M_{\mathcal{C}}$; для каждого сегмента задана функция кэширования $\mathfrak{R}_\sigma$; задана $F : \omega \to \sigma$ — ***функция выбора сегмента*** и $\mathfrak{S}$ такова: получается $\Sigma_{\mathfrak{S}}(t)$, затем вычисляются $\sigma = \mathrm{F}(\omega)$ и $E_\sigma(t)$, и состояние кэша после события ε равно: $\left(\cup_{\sigma \in \Sigma_{\mathfrak{S}}} \Psi_\sigma \in \mathfrak{R}_\sigma\left(\sigma,\ E_{\sigma\bar{\varepsilon}}\right),\ M\right)$.

Если для всех сегментов функции $\mathfrak{S}$ задана одинаковая функция $\mathfrak{R}$, то будем называть такую функцию $\mathfrak{S}$ ***основанной на функции*** $\mathfrak{R}$ и обозначать $\mathfrak{S}\left\langle\mathfrak{R}\right\rangle$.

Наконец, самое чёткое определение дадим той функции, на основе которой и будет строиться вся дальнейшая работа — ***атрибутивно-сегментная функция кэширования***.

Обозначим как $\mathrm{R}(\Omega)$ выбор случайного $\omega \in \Omega$.

Атрибутивно-сегментная функция кэширования — сегментная функция кэширования $\mathfrak{A} \in \{\mathfrak{S}\}$ такая, что для каждого $\sigma \in \Sigma_{\mathfrak{A}}$ заданы Ξ_σ — ***атрибуты сегмента*** и функция выбора сегмента задана следующим образом: если $\exists\sigma : \Psi_\sigma \ni \omega$, то $F(\omega) = \sigma$, иначе $\mathrm{F}(\omega) = \mathrm{R}\left(\tilde{\Sigma}\right)$, где $\tilde{\Sigma} = \cup_{\sigma \in \Sigma_{\mathfrak{A}}:\ \Xi_\sigma \subseteq \Xi_\omega,\ \nexists\tilde{\Xi}:\ \Xi_\sigma \subset \tilde{\Xi} \subseteq \Xi_\omega} \sigma$.

Для проведения аналитической части работы нужно ввести *качественную характеристику* функции кэширования. В качестве такой характеристики будет использоваться **BHR (byte hit-rate)** — отношение суммарного размера объектов, запросы к которым «попали в кэш» (т. е., в момент запроса объекта он находился

в кэше), к суммарному размеру всех запрошенных объектов. Суммарные размеры считаются с повторениями — т. е., разные запросы к одинаковым объектам считаются столько раз, сколько они встречаются в истории.

Таким образом, величина BHR будет равна:

$$\mathcal{B}\left(\mathfrak{R}, \mathcal{C}, E\right) = \frac{\sum_{\varepsilon=(Q\langle\omega\rangle,\ t)\in E:\ \delta_{\mathcal{C},\varepsilon}=\mathbf{true}} W_\omega}{\sum_{\varepsilon=(Q\langle\omega\rangle,\ t)\in E} W_\omega}$$

.

3.4 Генератор и эксперименты

Теперь введём необходимые определения для построения экспериментальной среды, в частности, генератора наборов данных. Как мы описали, генератор по заданным параметрам (сами параметры в определении никак не уточняются и будут описаны далее, при построении реализации) возвращает некоторую историю событий.

Итак, ***генератор*** — функция $\mathcal{G} : \tau,\ G \to \{E\}$, где τ — *таблица атрибутов*, $G = \{g\}$ — ***параметры генератора***, E — *история событий*.

Как видно из определения, по заданным параметрам может быть сгенерирован не один набор, а целое множество таких наборов (это будет важно для реализации). Поэтому в рамках программной реализации мы будем говорить о ***запуске генератора*** — выборе одного набора из этого множества: $\mathrm{R}\left(\mathcal{G}\left(\tau,\ G\right)\right)$ (или конечной серии таких запусков).

Конечно, главное, что нас интересует — не сам генератор или создаваемые им наборы данных, а *показатель BHR*, который получается при обработке кэш-функцией сгенерированного набора данных.

Определим ***серию из N экспериментов для заданных G, $\mathcal{G}$, τ, $\mathcal{C}$ и $\mathfrak{R}$*** как множество $E_N\left\langle\mathcal{G}\right\rangle\left(\mathfrak{R}, \tau, \mathcal{C}, G\right) = \left\{\mathcal{B}\left(\mathfrak{R}, \mathcal{C}, \mathcal{G}\left(\tau, G\right)\right)\right\}^N,\ |E_N| = N$.

Поскольку даже на одинаковых параметрах генератор может создать совершенно разные наборы данных, BHR на этих данных тоже может отличаться. Поэтому для определения качественной характеристики генератора (BHR на заданных параметрах) нужно вводить не одно число, а целый промежуток.

Введём ***диапазон исходов генератора*** как отрезок $D\left\langle\mathcal{G}\right\rangle\left(\mathfrak{R}, \tau, G\right) = \left(\underline{D},\ \overline{D}\right)$, где $\underline{D} = \lim_{N\to\infty} \min_{\mathcal{B}\in E_N\langle\mathcal{G}\rangle(\mathfrak{R},\tau,\mathcal{C},G)} \mathcal{B}$, $\overline{D} = \lim_{N\to\infty} \max_{\mathcal{B}\in E_N\langle\mathcal{G}\rangle(\mathfrak{R},\tau,\mathcal{C},G)} \mathcal{B}$.

Заметим, что всех возможных исходов генератора не более чем счётное количество, поэтому вышеописанный диапазон указывает, на границы не непрерывного сегмента или отрезка, а лишь счётного (или даже конечного) множества.

Проблема в том, что, теоретически, чтобы достоверно определить диапазон исходов генератора, потребуется бесконечная серия экспериментов. Поскольку работа носит прикладной характер, такое определение не имеет практического смысла. Поэтому вместо точного определения диапазона мы будем *оценивать его с некоторой вероятностью*.

Определим ***оценку диапазона исходов с вероятностью $p \in [0,\ 1]$ и точностью $\varepsilon \geqslant 0$*** как отрезок $D_{p,\varepsilon}\left\langle\mathcal{G}\right\rangle\left(\mathfrak{R}, \tau, \mathcal{C}, G\right) = \left(\underline{D}_p,\ \overline{D}_p\right)$, где $\underline{D}_p = \min_{\mathcal{B}\in E_N\langle\mathcal{G}\rangle(\mathfrak{R},\tau,\mathcal{C},G)} \mathcal{B}$, $\overline{D}_p = \max_{\mathcal{B}\in E_N\langle\mathcal{G}\rangle(\mathfrak{R},\tau,\mathcal{C},G)} \mathcal{B}$ и такой, что $\mathrm{P}\left(\left|\underline{D}_p - \underline{D}\right| \leqslant \varepsilon,\ \left|\overline{D}_p - \overline{D}\right| \leqslant \varepsilon\right) \geqslant p$. При этом число $\min_{E_N\langle\mathcal{G}\rangle(\mathfrak{R},\tau,\mathcal{C},G)} N$ будем называть ***минимальным числом экспериментов*** для такой оценки.

Далее потребуется всё же сделать одно уточнение о природе параметров генератора: области их определения должны быть **ограничены**.

Тем не менее, для поиска требуемого набора может потребоваться перебор *всех* возможных значений параметров. Однако, если хотя бы один из параметров представляет собой, например, вещественное число, находящееся даже в сколь угодно малом промежутке допустимых значений, полный перебор будет принципиально невозможен (поскольку такое множество будет несчётным). Поэтому вместо перебора всех возможных значений будет проверяться лишь их конечное число.

Для дальнейших уточнений потребуется ввести ещё одно промежуточное определение.

Будем говорить, что генератор $\mathcal{G}$ обладает ***гладкостью на заданной функции кэширования*** $\mathfrak{R}$, если $\exists\tau\forall\varepsilon > 0,\ \varepsilon \in \mathbb{R}\ \exists\delta > 0,\ \delta \in \mathbb{R} :\ \forall G_1, G_2 \in \{G\}\ \rho(G_1, G_2) < \delta \Rightarrow$

$$\frac{|D\langle\mathcal{G}\rangle(\mathfrak{R}, \tau, \mathcal{C}, G_1) \cap D\langle\mathcal{G}\rangle(\mathfrak{R}, \tau, \mathcal{C}, G_2)|}{|D\langle\mathcal{G}\rangle(\mathfrak{R}, \tau, \mathcal{C}, G_1) \cup D\langle\mathcal{G}\rangle(\mathfrak{R}, \tau, \mathcal{C}, G_2)|} > 1 - \varepsilon.$$

3.5 Постановка задачи

Введя все определения, можно, наконец, точно определить формальную постановку задачи:

1. Для заданной функции кэширования $\mathfrak{R}$ получить атрибутивно-сегментную функцию $\mathfrak{A}\langle R\rangle$, разработав для неё алгоритм вычисления $\sigma_{\mathfrak{A}}(t)$.

2. Для заданной таблицы атрибутов τ и полученной функции $\mathfrak{A}$ задать конечное множество наборов G параметров для заданного генератора $\mathcal{G}$.

3. Для каждого набора G из заданного множества и заданных $\mathfrak{A}$, $\mathfrak{R}$, $\mathcal{G}$, $\mathcal{C}$ и τ получить оценки $D_{\mathfrak{R}}(G) = D_{0{,}99,\ 0{,}01}\langle\mathcal{G}\rangle(\mathfrak{R}, \tau, \mathcal{C}, G)$ и $D_{\mathfrak{A}}(G) = D_{0{,}99,\ 0{,}01}\langle\mathcal{G}\rangle(\mathfrak{A}, \tau, \mathcal{C}, G)$.

4. Определить, существует ли такой набор G, что на этом наборе $\overline{D}_{\mathfrak{A}}(G) > \overline{D}_{\mathfrak{R}}(G)$.

4 Практическая часть. Разработка набора приложений

Поставленная задача будет решаться путём разработки программного комплекса.

Практическая часть будет состоять из двух основных разделов — построение собственно алгоритма (функции) кэширования и кэширующей системы, и разработка среды для проведения экспериментов.

Начальное исследование данных, подготовка их к обработке и некоторые детали реализации были проведены с использованием языка программирования Python 3.

Конечный программный комплекс разрабатывается на языке C++17 с использованием стандартных контейнеров и алгоритмов STL.

4.1 Описание и формат входных данных

Данные для моделирования процесса кэширования состоят из двух основных частей:

1. **Таблица атрибутов.** В таблице указаны атрибуты каждого объекта. Каждый атрибут имеет форму пары строк «Имя параметра — значение параметра» (например, `LANGUAGE='English'`);
2. **История запросов.** В ней последовательно перечислены запросы, для каждого из которых указан идентификатор объекта, его размер и время запроса.

Типы данных В рамках данной задачи в качестве базовых типов данных используются (здесь и далее пространство имён `std` считается подключенным):

- Идентификаторы объектов: строка (`string`);
- Один атрибут объекта: пара строк
  `typedef pair<string, string> TValuePair;`
- Множество атрибутов одного объекта:
  `typedef vector<TValuePair> TObjectAttr;`
- Таблица атрибутов:
  `typedef map<string, TObjectAttr> TAttrTable;`

4.2 Разработка алгоритма кэширования

Система кэширования, основанная на атрибутивном сегментировании, среди прочих параметров, должна иметь некоторую ***базовую стратегию*** (или, в терминах работы, *функцию кэширования*). Выбранный алгоритм используется для перенаправления запросов к выбранному сегменту. Система построена так, что и внешняя система, и функции кэширования сегментов (называемые далее ***подкэшами***) имеют единый интерфейс, что позволяет унифицировать многие элементы системы, что особенно актуально для тестирования модели и проведения экспериментов. Опишем эти детали подробнее.

4.2.1 Интерфейс кэша и требования к подкэшам

Любой кэш или подкэш для работы с системой должен обладать двумя основными интерфейсами: возможность запроса объекта, и возможность изменения размера (то есть, как и математическая функция кэширования, поддерживать соответствующие события). Для этого зададим интерфейс следующего вида:

```
class ICache{
public:
    virtual bool Get(string id, size_t size, time_t time) = 0;
    virtual void SetSize(size_t newSize) = 0;
};
```

Здесь результатом функции `Get` будет *результат запроса* соответствующего объекта (т. е., находился ли объект в кэше в момент, когда он был запрошен).

4.2.2 Метод временных слотов

Проверка наличия мотива для заданного набора атрибутов, по сути, сводится к анализу формы некоторой кривой, показывающей количество запросов к данному множеству объектов в момент времени. Очевидно, для проведения анализа эту кривую необходимо *дискретизировать* — то есть, поделить на некоторые фиксированные по величине части, которые станут единицей подсчёта количества запросов к данным.

Для дискретизации кривой числа запросов от времени была создана модель ***временных слотов***. Каждый слот представляет собой независимое хранилище статистической информации, которая далее будет использоваться для анализа. Набор слотов задаётся двумя параметрами — **длиной одного слота** и **количеством слотов**. Их произведение задаёт периоды потенциальных мотивов: система из N слотов, каждый из которых имеет длину t, может определять мотивы с длиной периода $v : Nt \vdots v$.

Далее слоты используются следующим образом:

1. Каждый поступающий запрос отправляется в один из слотов и используется для обучения системы;

2. В момент смены слотов происходит ***пересчёт мотивов***: очередной слот анализирует историю запросов к нему и возвращает информацию о мотивах — в частности, соответствующие атрибуты, размеры и т. п. Далее подсистема распределения сегментов строит *множество сегментов*, опираясь на эти данные.

4.2.3 Обработка очередного запроса, хранение истории

Слот хранит в себе статистические данные о поступивших запросах. Организованы они следующим образом:

- Заводится отдельная запись для каждого *мотива* — множества атрибутов (**не включая** пустое множество);
- Эта запись хранит в себе таблицу, в которой для каждого идентификатора содержится количество запросов к этому объекту;
- Каждый поступающий запрос на объект распределяется в одну или несколько таких записей;
- Записи с мотивами добавляются в реальном времени по мере необходимости.

Когда в слот попадает очередной запрос, он добавляется в историю всех мотивов, под которые он подпадает. Например, если атрибуты некоторого объекта были равны $\Xi = \{A, B, C\}$, то он попадёт в историю мотивов $\{A\}$, $\{B\}$, $\{C\}$, $\{A, B\}$, $\{A, C\}$, $\{B, C\}$, $\{A, B, C\}$.

Упрощённо структуру одной записи-мотива можно описать так:

```
typedef map<TObjectAttr, map<string, size_t>> TMotive;
```

4.2.4 Подсчёт мотивов

При смене слота будущий активный слот предоставляет набор ***потенциальных мотивов***, на основании которого будет производиться сегментирование кэша.

Потенциальный мотив формируется на основании истории запросов, поступивших во временной слот за все циклы. Основу алгоритма составляет *подсчёт объёма трафика*, поступившего в слот за разные циклы; дополнительно могут применяться различные улучшения, такие как отсечение по пороговым значениям, сглаживание пиков, коэффициенты устаревания и т. п.

Потенциальный мотив представляет собой пару «набор атрибутов — относительный размер». Алгоритм вычисления потенциальных мотивов очень простой:

1. Мотивы сортируются по убыванию (невозрастанию) ***качества мотива*** (см. далее);

2. Выбираются первые несколько мотивов в соответствии с ***правилом отсечения*** (см. далее).

Качество мотива — некоторая численная величина, которая используется для сравнения различных мотивов между собой. Этот показатель может быть различным, причём от выбранного показателя в том числе зависит конечный показатель всего кэша; в рамках задачи в качестве показателя используется **доля трафика, соответствующего мотиву среди всего трафика за данный промежуток времени (слот)**.

Правило отсечения — правило, определяющее, какие из найденных мотивов попадут в итоговое распределение, а какие будут отброшены; в рамках задачи используется правило отсечения по двум значениям: **минимальное качество мотива** и **максимальное количество мотивов**. Эти параметры, наряду с остальными, задаются при запуске системы.

Размер нового сегмента в рамках данного показателя задаётся как произведение качества мотива на долю объектов без повторений, входящих в мотив, к доле всех поступивших объектов (*размер сегмента* здесь задаётся как относительная доля от общего размера кэша).

4.2.5 Перераспределение сегментов, заполнение данными

Когда система получает новый набор мотивов, она начинает распределять сегменты кэша в соответствии с этими мотивами. В представленной версии алгоритм распределения сегментов устроен следующим образом:

Обозначим текущее множество сегментов Σ, а полученное множество потенциальных мотивов — V, причём сегмент представляет собой тройку $\sigma = (\Xi_\sigma, M_\sigma, \Psi_\sigma)$, где Ξ_σ — атрибуты сегмента (мотива), M_σ — размер сегмента, Ψ_σ — множество объектов, содержащихся в сегменте, а мотив — пару $v = (\Xi_v, M_v)$. Новое множество сегментов, которое будет получено после перераспределения, обозначим Σ'.

1. Если для данного $\sigma \in \Sigma\ \exists v \in V : \Xi_v \subseteq \Xi_\sigma$, то к нему применяется операция $S\langle M_v \rangle$, после чего $\Sigma' \ni \sigma' = (\Xi_v, M_v, \Psi'_\sigma)$, где Ψ'_σ — объекты сегмента после выполнения указанной операции.

2. В противном случае $\sigma \notin \Sigma'$.

3. Если для данного $v \in V\ \nexists \sigma \in \Sigma : \Xi_v \subseteq \Xi_\sigma$, то $\Sigma' \ni \sigma_v = (\Xi_v, M_v, \mathrm{P}(\Xi_v, M_v))$, где $\mathrm{P}(\Xi, M)$ — ***предзагрузка объектов по атрибутам*** Ξ ***размером*** M:

функция, которая по заданным атрибутам получает множество объектов, обладающих этими атрибутами, таких, что их суммарный размер не превосходит M. Принцип, по которому выбираются эти объекты, зависит от реализации.

4. Далее для всех $\omega : \exists\sigma_1, \ldots, \sigma_n \in \Sigma' : \sigma_1 \neq \ldots \neq \sigma_n\omega \in \Psi_{\sigma_1} \ldots \omega \in \Psi_{\sigma_n}$ выбирается $\sigma_{\max} = \text{argmax}_{\sigma\in\Sigma'} |\Xi_\sigma|$ и далее $\Psi'_\sigma := \Psi'_\sigma \setminus \omega \; \forall\sigma \neq \sigma_{\max}$.

4.2.6 Общая схема системы

В целом, схему работы системы можно представить следующим образом:

1. Поступивший от клиента запрос обрабатывается сегментами: если в одном из них находится объект, он возвращается клиенту; при этом поступает запрос к соответствующему сегменту;

2. Если ни в одном сегменте нет запрашиваемого объекта, определяется, какому сегменту, согласно их атрибутам, он должен находиться, и производится запрос к этому сегменту (далее сегмент действует согласно своей стратегии);

3. Запрос добавляется в историю текущего временного слота;

4. Если на очередном запросе происходит смена временного слота, из системы анализа запрашиваются новые мотивы на текущий временной слот;

5. Сегменты кэша перераспределяются в соответствии с полученными мотивами.

Графически схема работы системы приведена на рис. 1.

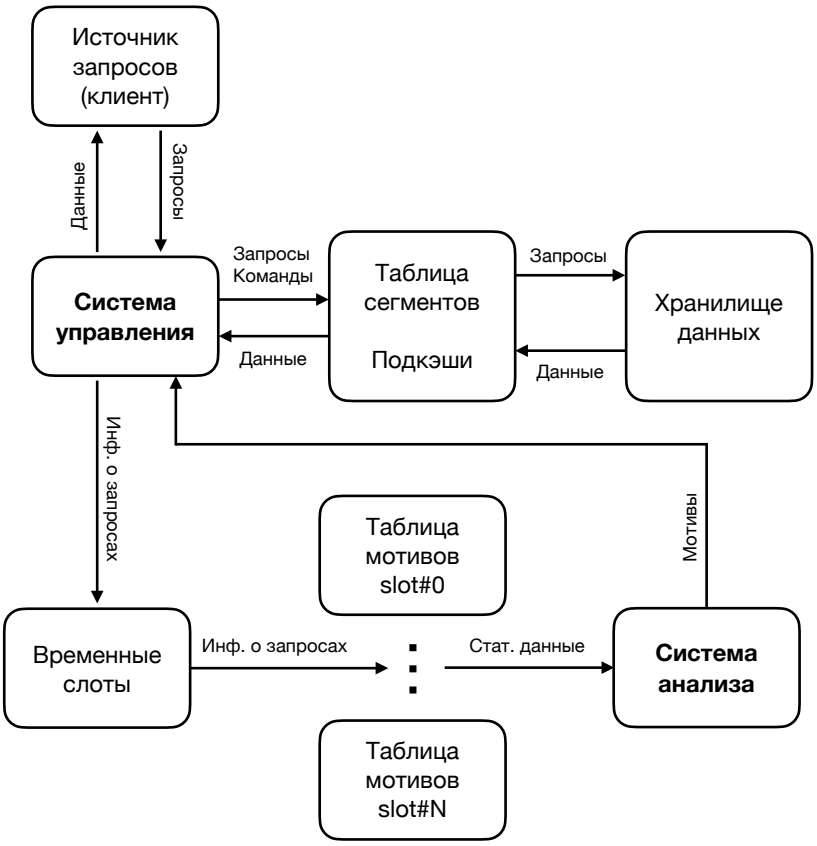

Рис. 1: Схема взаимодействия частей системы кэширования

4.3 Экспериментальный комплекс

Среда для проведения экспериментов будет состоять из двух основных частей: симулятора кэша и генератора данных.

4.3.1 Симулятор кэша

Задача симулятора кэша — смоделировать заданное множество событий на заданной кэш-системе (кэш-функции) и вычислить в качестве результата величину BHR.

Симулятор кэша просто считывает построчно указанный файл с историей событий и последовательно выполняет запросы к указанным объектам.

4.3.2 Генератор наборов данных

Для проведения экспериментов, помимо симулятора, потребуется множество наборов входных данных.

Параметры генератора Исходные данные для генератора задают конкретные мотивы и отдельно параметры каждого мотива. Для каждого мотива задаются,

во-первых, его атрибуты, а, во-вторых, характер распределения числа запросов во времени.

Полный список поддерживаемых параметров выглядит так:

- **Период мотива.** Показывает, с какой периодичностью будет встречаться мотив в наборе данных. Задаётся в единицах времени (напр., 1 неделя);
- **Длительность мотива.** Задаёт, сколько будет длиться одна итерация мотива (включая рост и угасание). Задаётся в единицах времени (напр., 3 часа);
- **Объём трафика мотива.** Задаёт суммарный объём трафика, соответствующего мотиву, за одну итерацию. Измеряется в единицах объёма (напр., 10 Гб);
- **Скорость роста и угасания мотива.** Задаёт, насколько плавно будет нарастать и спадать плотность трафика мотива. Измеряется в единицах времени (напр., 1 час — это будет время, за которое плотность трафика возрастает с нуля до максимальной величины, и за столько же он будет постепенно уменьшаться);

Параметры могут задаваться не точечно, а в виде диапазонов — на каждой итерации будет случайным образом выбираться некоторое число из этого диапазона.

Пример задания одного мотива для генератора:

```
[GENRE='Thriller', COUNTRY='USA']
period = 1w ; период: 1 неделя
length = 3h~4h ; длительность: 3-4 часа
shift = 1h ; сдвиг от начальной точки отсчёта
volume = 10G ; объём трафика: 10 Гбайт
attack = 20m~30m ; рост и спад: 20-30 минут
```

В одном файле может быть задано множество мотивов, включая так называемый ***пустой мотив*** — секцию с пустым множеством атрибутов `[]`, которая будет соответствовать всем объектам из таблицы.

Генератор последовательно создаёт историю запросов для каждого мотива, после чего объединяет их в единый конечный файл с общей историей запросов, упорядоченной по времени.

Запуск генератора включает в себя два входных параметра — файл конфигурации и *общее число запросов*, которые будут созданы. В результате запуска получается файл с историей запросов, который далее используется для запуска симулятора кэша.

Реализация генератора Поскольку генератор является в каком-то смысле обратной программой к системе анализа мотивов, в нём можно применить тот же подход для генерации наборов — *дискретизацию по временным слотам.*

Алгоритм работы такого генератора описывается довольно просто:

1. Выбираем очередной мотив для текущего слота;
2. Определяем, какой объём данных из этого мотива требуется в данный временной слот (согласно данной конфигурации);
3. Добавляем в итоговый набор случайные объекты суммарным объёмом не более этого объёма;
4. Переходим к следующему мотиву текущего слота или первому мотиву следующего слота.

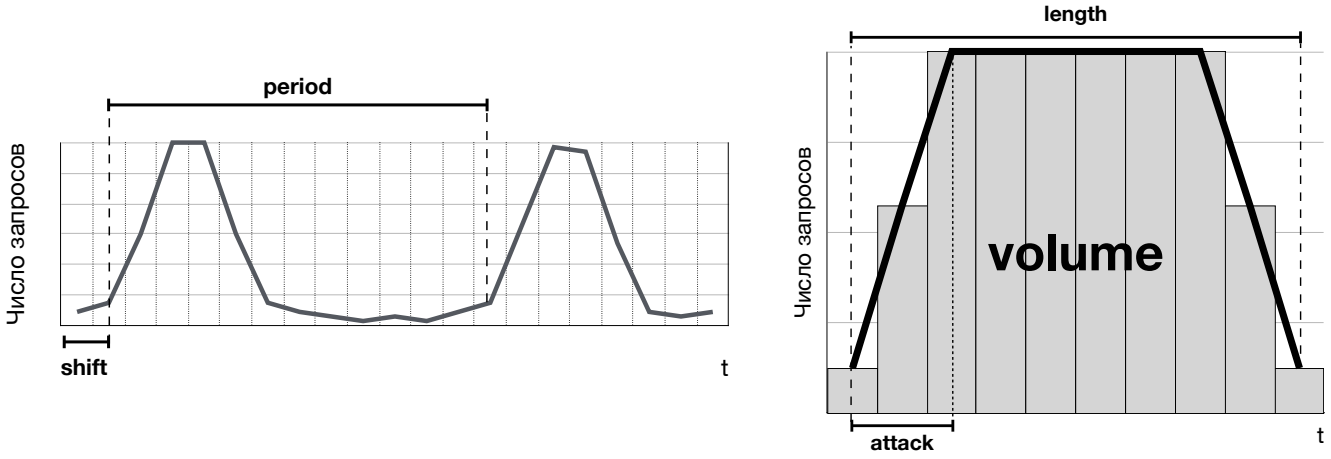

Рис. 2: Визуализация параметров генератора и процесса дискретизации

При этом есть некоторые особенности наборов, создаваемых таким генератором.

Во-первых, заполнение будет происходить не идеально по заданному размеру, а лишь приближённо: это связано с тем, что далеко не всегда можно подобрать множество объектов, в сумме дающих строго заданный объём. Поэтому генератор устроен так, что суммарный объём объектов лишь *не превосходит* заданный, а не равен ему в точности.

Во-вторых, для лучших результатов желательно выбирать частоту дискретизации (длину слота) генератора либо равной длине слота системы кэширования, либо такой, чтобы слот кэша имел длину, кратную длине слота генератора.

5 Практическая часть. Эксперименты

Для второго, экспериментального, раздела практической части работы будут в паре использоваться два программных средства — симулятор и генератор наборов.

5.1 Описание экспериментов

Одна серия экспериментов производится следующим образом:

1. Проектируется тестируемое распределение запросов и подготавливается соответствующий конфигурационный файл для генератора наборов данных;
2. Генератор запускается несколько (зависит от типа эксперимента, см. далее) раз на этом конфигурационном файле;
3. На полученных наборах данных запускаются последовательно или параллельно симулятор кэша LRU и симулятор атрибутивно-сегментного кэша **с одинаковым суммарным размером кэша**; при этом для атрибутивно-сегментного кэша может производиться несколько запусков на каждом наборе с разными параметрами алгоритма кэширования (см. Приложение) с последующим выбором из них лучшего результата;
4. Результатом каждого запуска симулятора является итоговое значение BHR;
5. Сравниваются значения BHR базового и предложенного алгоритма; положительным результатом эксперимента считается такой, у которого BHR предложенного алгоритма выше такового для базового.

5.1.1 Параметры генератора

Для создания наборов данных будут использоваться конфигурационные файлы разных уровней сложности.

Во всех случаях создаваемый набор данных будет иметь размер 1 000 000 запросов.

Частота дискретизации генератора установлена в 15 минут, а параметры слота алгоритма кэширования подбираются кратно длинам мотивов.

Остальные параметры алгоритма подбираются по возможности оптимальным образом для достижения наибольшего BHR. Размер кэша выбирается таким образом, чтобы итоговый BHR находился в пределах 0,20–0,80 (кроме последнего набора данных).

К каждому подразделу также приведены ***графики BHR***, показывающие величину BHR после обработки первых Rq запросов указанного набора данных. На графиках серым цветом обозначен базовый алгоритм (LRU без сегментирования), а чёрным — атрибутивно-сегментный алгоритм.

5.1.2 Легко разделяемые данные

Легко разделяемыми данными будем называть такие данные, в которых каждый временной слот на 100% занят данными ровно одного мотива. Например, цикл всех мотивов составляет 24 часа, из которых каждые 6 часов полностью заняты данными одного из четырёх заданных мотивов. Все мотивы соответствуют различным значениям **одного и того же** атрибута, причём каждый объект обладает не более чем одним значением этого атрибута. Такие мотивы гарантированно будут распознаны алгоритмом и именно на них, предположительно, может быть приблизительно оценен максимальный прирост алгоритма.

Пример одиночного запуска:

- Простой LRU — 0.503
- Атрибутивно-сегментный LRU — 0.703

Результат эксперимента: **положительный**.

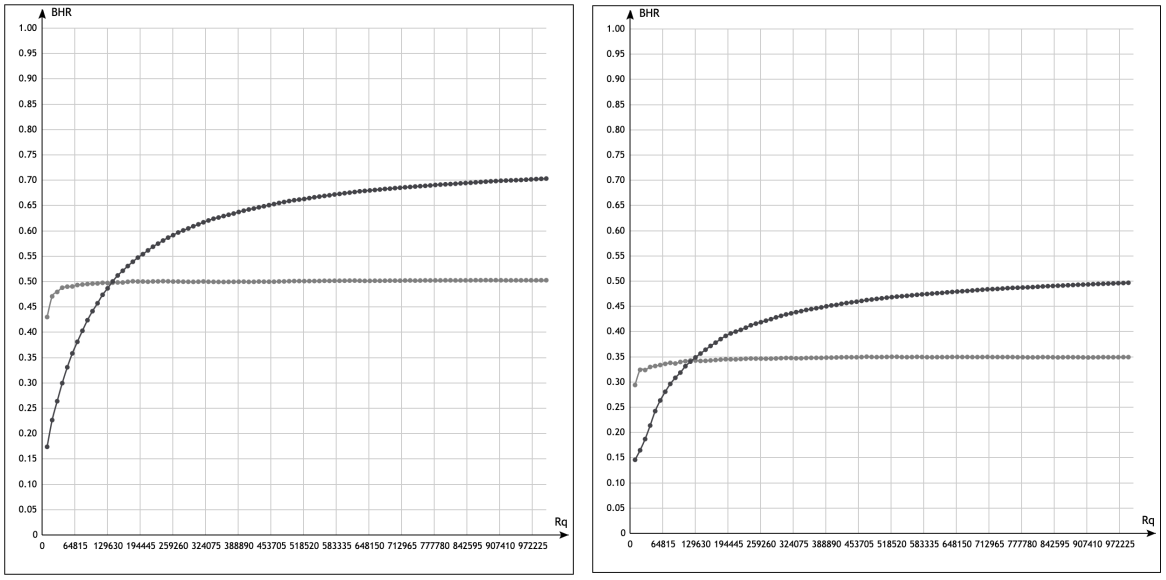

Рис. 3: Легко разделяемые данные (без шума и с шумом соответственно).

5.1.3 Легко разделяемые данные с шумом

Несколько усложнённый вариант предыдущего типа наборов. В этом случае, помимо данных, соответствующих одному мотиву, к ним добавляется постоянный шум из случайных объектов, не обладающих каким-либо мотивом. Эта серия экспериментов проверяет способность алгоритма находить лучший из обнаруженных мотивов.

Пример одиночного запуска:

- Простой LRU — 0.349
- Атрибутивно-сегментный LRU — 0.499

Результат эксперимента: **положительный**.

Можно заметить, что, хотя абсолютный BHR снизился в обоих случаях из-за возросшего абсолютного объёма трафика, в целом сохранились форма кривых, а также относительный прирост BHR улучшенного алгоритма, который составляет около $\frac{0.499}{0.349} \approx 40\%$.

5.1.4 Смешанные мотивы

В следующей серии наборов данных каждому временному слоту соответствует не один мотив, а несколько, причём в разные моменты времени соотношение между объёмами каждого из мотивов будет отличаться. Эта серия проверяет способность алгоритма работать более чем с двумя сегментами значимого размера (порядка десятков процентов от общего объёма кэша), а также правильно динамически вычислять оптимальный размер каждого из таких сегментов.

Пример одиночного запуска:

- Простой LRU — 0.517
- Атрибутивно-сегментный LRU — 0.677

Результат эксперимента: **положительный**.

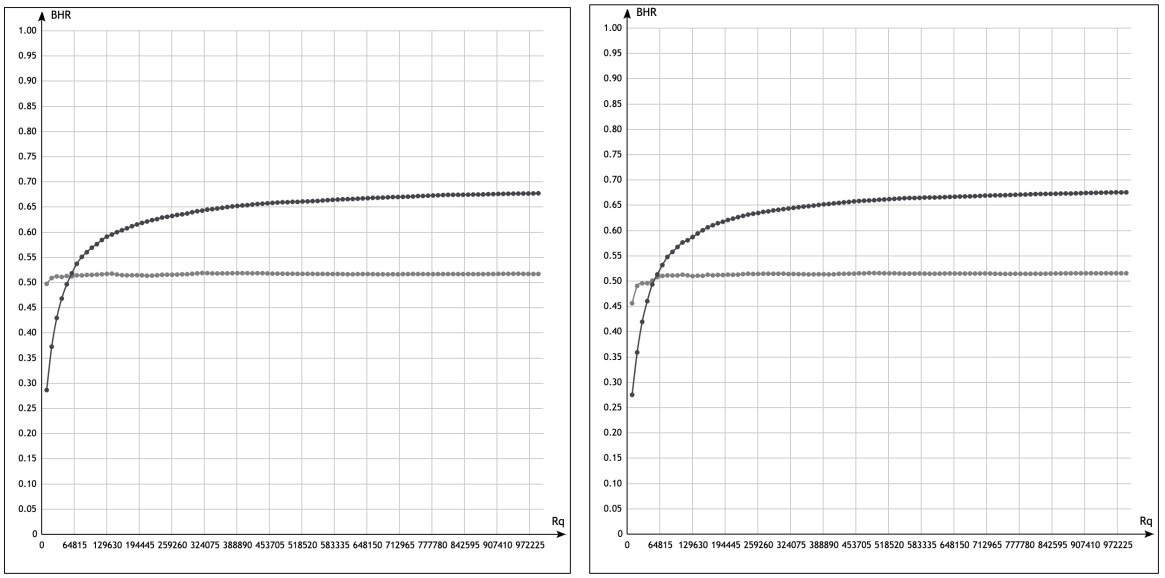

Рис. 4: Данные со смешанными мотивами (без шума и с шумом соответственно).

5.1.5 Смешанные мотивы с шумом

Эта серия отличается от предыдущей добавлением случайного шума к заданным мотивам, причём количество шума также непостоянно на протяжении времени.

Пример одиночного запуска:

- Простой LRU — 0.516
- Атрибутивно-сегментный LRU — 0.677

Результат эксперимента: **положительный**.

В этой серии наборов относительный прирост немногим меньше предыдущей — ок. 30%, однако форма кривых также сохраняется независимо от наличия шума в потоке данных.

5.1.6 Сложные данные с шумом

Последняя, самая высоконагруженная серия экспериментов. В этой серии в каждый момент времени присутствуют несколько разных мотивов и случайный шум, причём периоды различных мотивов не равны между собой.

Это самый сложный пример, наиболее приближенный к реальному поведению клиентов в сети (при условии, что это поведение соответствует гипотезе существования мотивов).

Пример одиночного запуска:

- Простой LRU — 0.239
- Атрибутивно-сегментный LRU — 0.257

Результат эксперимента: **положительный**.

Как видно из результата, самые сложные данные ожидаемо наиболее трудны для алгоритма, тем не менее, он всё ещё показывает стабильный прирост, но в данном случае лишь на 7,5%.

5.1.7 Случайный шум

Серия экспериментов для проверки наличия мотивов в абсолютно случайных данных. Конфигурационный файл в этой серии состоит из одного пустого мотива, у которого период и длина мотива равны, длина атаки равна нулю, а объём задан постоянной величиной.

Пример одиночного запуска:

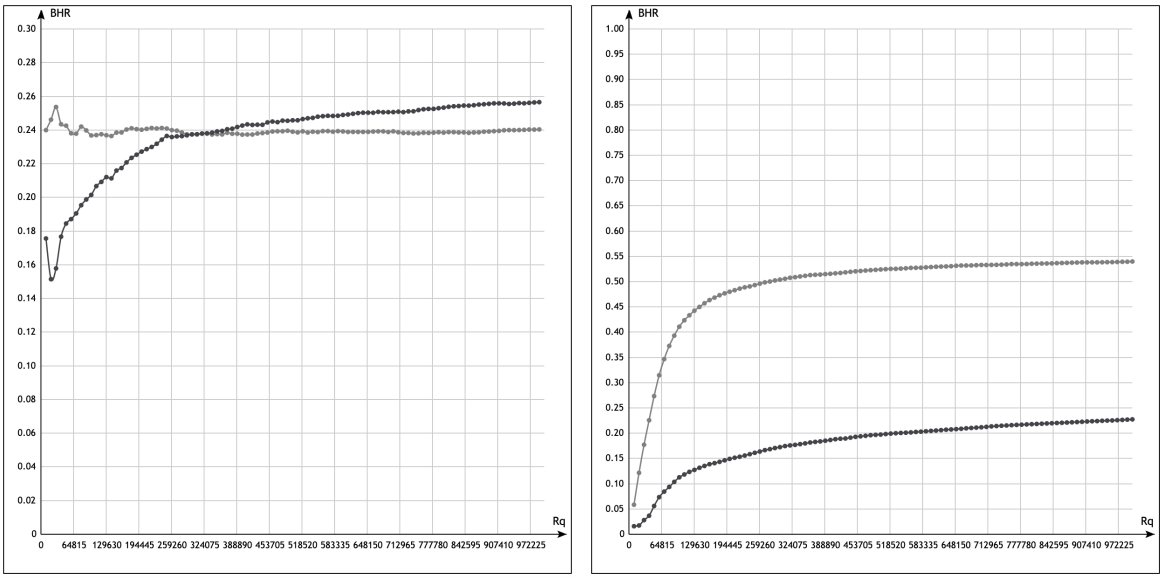

Рис. 5: Сложные данные с шумом и случайные данные соответственно.

- Простой LRU — 0.054
- Атрибутивно-сегментный LRU — 0.023

Результат эксперимента: **отрицательный**.

Как и ожидалось, в случайных данных алгоритм не смог найти ни одного мотива, и даже понизил результат в попытках их вычислить. Это значит, что применение алгоритма на неподходящих данных даёт даже худший результат, чем несегментированный алгоритм.

5.1.8 Примечание о числе запусков

Как было замечено в формальной постановке задачи, генератор выдаёт разный результат при каждом запуске на одинаковых входных параметрах, а значит, и BHR соответствующего эксперимента будет отличаться, поэтому требовалось найти *вероятную верхнюю оценку* BHR с погрешностью не более 0,01 (1%).

Для проверки разброса значений были проведены 100 тестовых запусков генератора и подсчёт разброса BHR на обоих алгоритмах. В результате этих тестов разброс значений составил **не более 0,3%**, что позволяет с достаточной уверенностью утверждать, что BHR для одних и тех же параметров генератора **совпадает с точностью до 1%**.

Для большей уверенности, каждый из конфигурационных файлов запускался на генераторе 3 раза. По итогам экспериментов разброс также не превысил 0,3%.

5.2 Результаты экспериментов

В результате вышеописанных экспериментов для входных данных разной сложности показан прирост BHR на всех наборах данных, кроме случайного на-

бора. Относительный прирост BHR составил от 7,5% до 42% в зависимости от серии наборов данных.

Поскольку задачей работы являлось определение существования *хотя бы одного* набора данных, показывающего положительный результат эксперимента, задача выполнена успешно, положительный результат показан.

Особенность алгоритма Если рассмотреть вышеприведённые графики BHR, на всех наборах, кроме случайного, можно заметить следующую особенность: сначала некоторое время BHR базового алгоритма превосходит BHR улучшенного, но затем последний заметно улучшает показатели. Это может быть связано с тем, что на малой выборке система обучается поначалу неверно, но затем корректирует данные и далее показывает лучший результат.

6 Результаты работы

В результате работы было показано, что, действительно, существуют такие наборы данных, на которых разработанный алгоритм кэширования показывает лучший показатель BHR, чем лежащий в его основе базовый алгоритм.

Также в целом подтвердилась гипотеза, что этот алгоритм показывает прирост именно на данных, в которых наблюдаются *мотивы*: на случайных данных, в которых мотивы отсутствуют, алгоритм показывает результат ниже базового алгоритма.

Более полное исследование, включающее определение доли положительных исходов эксперимента на всём пространстве входных данных и поиск границ признаков, позволяющих получить положительный результат, выходит за рамки данной работы, однако, предположительно, для текущей версии алгоритма пригодны данные, обладающие мотивами с периодом, являющимся делителем заданного периода алгоритма (т. о. оптимально выбирать период алгоритма как НОК всех мотивов, которые, предположительно, могут находиться в этих данных), а все остальные признаки — такие, как относительный объём трафика каждого из мотивов, наличие фонового шума, не обладающего мотивами и другие описанные выше признаки либо не влияют на работу алгоритма в заметной степени, либо в любом случае сохраняют ненулевой прирост алгоритма относительно базового.

7 Заключение

Хотя в данной работе экспериментальная часть строилась на наборах данных из CDN-сети, сфера применения алгоритма не ограничивается видеофайлами.

Как было описано, алгоритм применим практически к любым примерам данных, которые обладают какой-либо информацией, которую можно принять за *атрибуты*, например:

- У видеофайлов — информация из CDN, как указано выше;
- Для инструкций процессора — идентификатор процесса;
- Для сетевого пакета — например, правила обработки, идентификатор приложения и т. п.

Возможности для улучшения Уже сейчас можно назвать направления, в которых можно улучшить созданный алгоритм:

- **Динамическое вычисление периода мотива.** Сейчас периоды определяемых мотивов задаются фиксированным параметром, который подбирается эмпирическим путём исходя из естественного характера входных данных;
- **Разные периоды для мотивов.** Опять же, в нынешней версии, период определяемых мотивов одинаковый для всех сегментов (точнее, алгоритм может определять мотивы с периодом, являющимся **делителем** заданного параметра). Например, если длиной периода задана 1 неделя, то алгоритм может определять мотивы периодом в 1 неделю или, например, 1 день, но не определит мотив с периодом в 3 дня. С возможностью динамического определения периода мотива этот период можно подбирать разным для каждого мотива; для этого потребуется некоторое усложнение системы временных слотов;
- **Автоматическая фильтрация атрибутов.** В рамках работы для проведения экспериментов была проведена предварительная фильтрация входных данных с целью отсечения *мусорных атрибутов* — т. е., таких, которые гарантированно не несут полезной информации (например, технические данные файла, такие, как формат видеоконтейнера, системный идентификатор файла и т. п.). Можно дополнить алгоритм системой автоматической фильтрации таких данных (например, исключать из поиска атрибуты, которые какое-то время не образуют мотива);

Список литературы

1. Podlipnig S., Böszörmenyi L. A survey of Web cache replacement strategies. // ACM Computing Surveys. 2003. N 35. P 374–398.

2. De Vleeschauwer D., Robinson D. Optimum Caching Strategies for a Telco CDN. // Bell Labs Technical Journal. 2011. N 16. P 115–132.

3. Huang Q., Birman K., Van Renesse R., Lloyd W., Sanjeev K., Li H. C. An analysis of Facebook photo caching. // SOSP '13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013. P 167–181.

4. Figueiredo F., Almeida J., Gonçalves M., Benevenuto F. TrendLearner: Early Prediction of Popularity Trends of User Generated Content. // Information Sciences. 2014. N 349.

5. Berger D. S., Sitaraman R. K., Harchol-Balter M. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network. // NSDI'17: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation. 2017. P 483–498.

A Приложение

A.1 Исходные коды программного комплекса

Исходные коды приложений доступны в репозитории GitHub: `https://github.com/himotokun/scache`.

A.2 Тестовые наборы данных для экспериментов

Находятся в том же репозитории, что и исходные коды, в подкаталоге `/examples`. В нём находятся:

- Конфигурационные файлы, которые использовались для запуска генератора;
- Примеры готовых наборов данных для запуска симулятора;
- Таблица соответствия атрибутов и объектов (в формате JSON).

A.3 Руководство по сборке и запуску приложений

Сборка исходного кода производится в автоматическом режиме командой `make`.

В результате сборки исходных кодов получаются три приложения — `scache`, `lru` и `generator`. Первые два приложения — симуляторы соответственно атрибутивно-сегментного кэша и простого кэша LRU. Третье приложение — генератор наборов данных.

Симуляторы требуют для запуска указания параметров в командной строке; полное их описание можно получить, запустив приложения без параметров.

Генератор может работать в трёх режимах: одиночном, пакетном и интерактивном:

- В *одиночном* режиме генератор создаёт один набор данных на основе конфигурационного файла, указанного в параметрах командной строки;

- В *пакетном* режиме генератор создаёт множество наборов данных на основе ***пакетного файла***, путь к которому указывается в параметрах командной строки;

- В *интерактивном* режиме все операции производятся путём ввода команд в командную строку генератора (список команд выводится при запуске или вводе команды `help`); запуск в интерактивном режиме производится без параметров командной строки.

Справку по режимам работы и параметрам командной строки можно получить, запустив приложение с параметром `help` (напр., `./generator help`).