САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Крайчик Георгий Ильич

# Восстановление параметров потока обращений к устройству хранения

Курсовая работа

Научный руководитель:
профессор Нестеров В. М.

Санкт-Петербург
2014

# Оглавление

# Введение

Важнейшей задачей организации работы системы хранения данных является увеличение скорости доступа к информации. Существуют различные способы решения данной задачи. Одним из способов является приобретение высокоскоростного оборудования, существенно увеличивающего общую производительность системы, например, оборудования на основе SSD устройств хранения. Однако данное решение является крайне дорогостоящим из-за потребности хранения чрезмерно больших объемов данных, что приводит к выводу об экономической нецелесообразности такого подхода.

Оптимальным решением проблемы ускорения работы системы хранения является оснащение ее кэш-памятью. Эффективность использования кэш-памяти базируется на двух аспектах:

1. выбор оптимального размера кэш-памяти таким образом, чтобы достичь оптимального соотношения между затратами и повышением производительности системы

2. разработка алгоритмов, позволяющих наилучшим образом управлять заполнением и освобождением кэша

Различные системы хранения обладают различными вычислительными мощностями. Крупные системы хранения обладают отдельно выделенным устройством управления кэш-памятью, в то время как архитектура мелких систем хранения не подразумевает наличие такового. Отсюда следует, что трудоемкие с точки зрения вычислительной сложности алгоритмы будут чрезмерно загружать процессор мелких систем хранения, что приведет к общему падению производительности этих систем. Это означает, что при разработке алгоритма управления кэш-памятью нужно учитывать не только его вычислительную сложность, но и тип системы хранения, под которую он разрабатывается.

Сущетвует несколько различных принципов построения алгоритов управления кэш-памятью. Для тех систем хранения, для которых не выявлено никаких статистических закономерностей в распределении потока обращений, применяются алгоритмы LRU, MRU, LFU и AFC. В то же время для более сложных систем хранения данных применяются алгоритмы, основывающиеся на анализе собираемой статистической информации о загрузке системы хранения.

В настоящей работе разрабатывается алгоритм управления кэш-памятью для систем хранения данных, использующий информацию о характере распределения потока обращений. По мониторингу потока ввода-вывода информации требуется выявить параметры распределения потока обращений к устройству хранения с целью последующего предсказания загрузки системы хранения данных.

# 1. Принцип работы кэш-памяти

Кэш - это высокоскоростная статическая энергозависимая память. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной оперативной памяти или жесткого диска. Является очень дорогостоящей памятью, поэтому чрезмерное ее использование ведет к росту стоимости вычислительных систем. В связи с этим, кэш используется в небольших объемах с целью повышения производительности доступа к данным. Чтобы повысить скорость доступа к данным, имеет смысл размещать в кэш-памяти те данные, вероятность обращения к которым в ближайшем будущем максимальна.

Из основной памяти данные считываются блоками фиксированной длины (страницами), поэтому данные добавляются и извлекаются из кэша также страницами. В каждый момент времени система управления кэш-памятью должна уметь определять, какие страницы загружены в кэш, а какие нет. Для этого в системе управления кэшем заводится отдельный участок памяти (далее называемый стеком), в котором хранится список адресов страниц основной памяти, которые были загружены в кэш. Между ячейками стека и кэш-памятью имеется взаимнооднозначное соответствие. Таким образом, каждому элементу стека поставлена в соответствие единственная страница кэша и наоборот. Общий принцип работы системы управления кэш-памятью изображен на Рисунке 1.

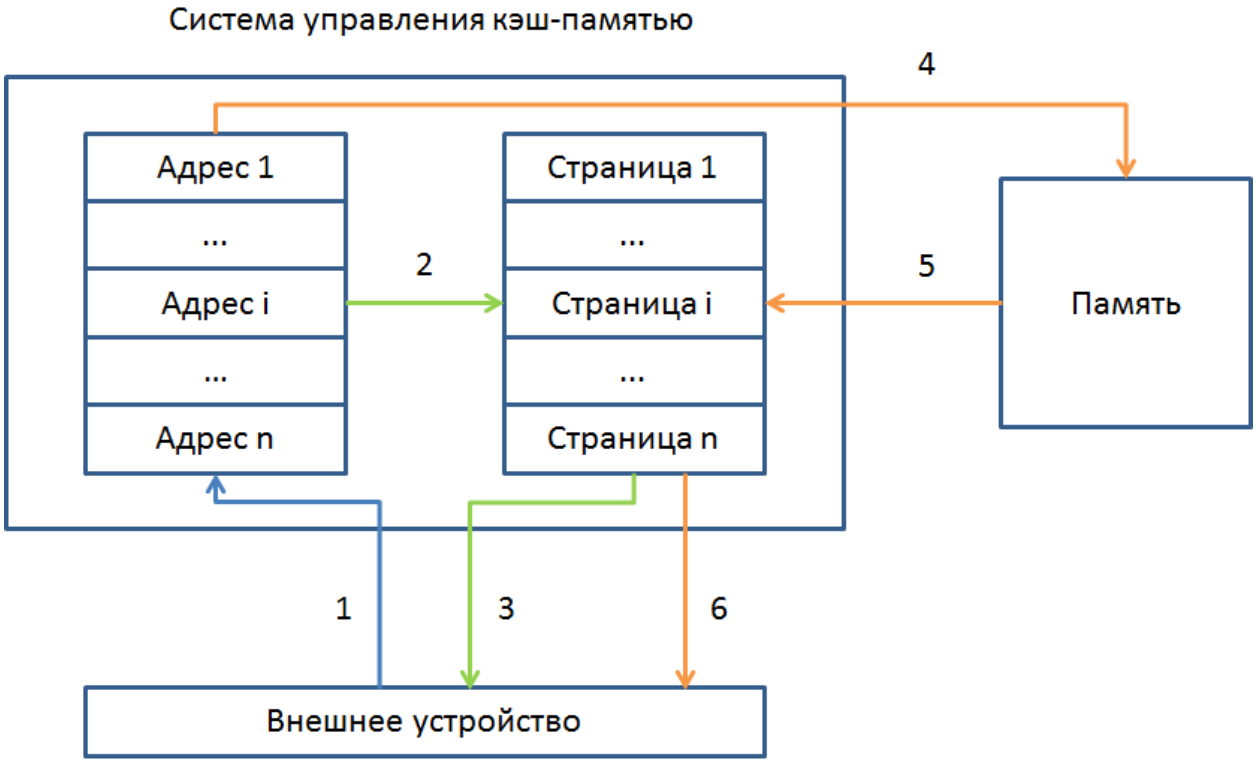


Рис. 1: Управление кэш-памятью

1. От внешнего устройства приходит очередной запрос на чтение данных из памяти по некоторому адресу addr. Системе управления кэш-памяти требуется определить, загружены ли данные, лежащие по адресу addr, в кэш. Для этого осуществляется поиск addr в стеке.

2. Если addr был найден в стеке, говорят, что произошел cache-hit. В этом случае чтение данных, лежащих в основной памяти по адресу addr, будет осуществляться из кэша.

3. Данные, прочитанные из кэша, передаются внешнему устройству. Следует обратить внимание, что при обработке запроса 1-2-3 основная память не была задействована. Именно этот факт и повышает скорость доступа к данным.

4. Если addr не был найден в стеке, то говорят, что произошел cache-miss. Это означает, что необходимые данные отсутствуют в кэше и что их требуется загрузить напрямую из памяти.

5. При появлении новых данных в кэше специальный алгоритм вытеснения определяет необходимость сохранения новых данных в кэше с дальнейшим перенаправлением этих данных внешнему устройству. Алгоритм вытеснения данных является ключевым для определения набора данных, подлежащих хранению в кэше. Производительность системы управления кэш-памятью напрямую зависит от эффективности данного алгоритма.

6. Считанные из основной памяти данные передаются внешнему устройству.

Заметим, что до тех пор пока кэш не заполнен полностью, имеет смысл продолжать его заполнять. Однако после того, как кэш оказывается заполненным полностью, требуется принять решение, какие страницы в нем оставлять, а какие извлекать. Принятием подобных решений занимаются специальные вытесняющие алгоритмы.

Существует несколько подходов к вытеснению данных их кэша. В их основе лажат три различные концепции:

1. Вытесняется та страница, которае не использовалась дольше всех. Основывается на предположении, что чем дольше не использовались данные, тем меньше вероятность того, что они вновь пригодятся. Данную идею воплощает в себе LRU-алгоритм (Least Recently Used).

2. Вытесняется последняя использованная страница. Эта концепция является абсолютной противоположностью предыдущей, так как основывается на предположении, что чем дольше не использовались данные, тем больше вероятность того, что вскоре они понадобятся. Данная идея была реализованы в MRU-алгоритме (Most Recently Used).

3. Вытесняется та страница, которая используется реже всех. Данный подход был реализован в LFU-алгоритме (Last Frequently Used).

Далее будет подробно описана работа LRU-алгоритма вытеснения данных.

# 2. Описание работы алгоритма LRU

Пусть M - количество страниц, помещающихся в кэше. Пусть N - количество запросов к системе хранения данных. $\{x_i\}_{i=1}^{N}$ - последовательность адресов (запросов).

Опишем работу LRU-алгоритма. Пусть на вход устройству пришел очередной адрес $x_j$, где $1 \leq j \leq N$. Последовательно перебираем адреса, находящиеся на стеке. Как только найдется $i$, такой что $x_i = x_j$, извлекаем $x_i$ из его текущего положения в стеке и кладем его на вершину. При этом стековым расстоянием адреса $x_j$ назовем число $i - 1$. В случае, если такого $i$ не нашлось, сдвинем весь стек на 1 элемент вниз (тем самым возможно вытеснение адреса $x_M$), а на вершину стека запишем адрес $x_j$. При этом стековым расстоянием адреса $x_j$ будем считать $+\infty$.

Стековые расстояние играют важную роль в задаче восстановления параметров потока обращений. Известно, что в большинстве практических ситуаций стековые расстояния последовательности адресов распределены по закону Парето. Таким образом, чтобы восстанавливать параметры распределения адресов необходимо уметь вычислять стековые расстояния максимально быстро. Вычислительную сложность описанной выше реализации алгоритма LRU можно оценить как $O(N \cdot M)$, что не достаточно эффективно. В связи с этим, возникает потребность в разработке новой реализации алгоритма LRU, которая бы работала эффективнее, чем $O(N \cdot M)$.

Предлагается эмулировать работу стека при помощи интервального АВЛ-дерева.[2] В нем будут храниться номера тех адресов, которые в данный момент в стеке отсутствуют. После чтения очередного адреса $x_j$, будем изменять структуру дерева, при этом поддерживая его сбалансированность.

В узлах дерева будут храниться не одиночные адреса, а целые интервалы адресов вида [a, b], которые в данный момент времени на стеке отсутствуют. У каждого узла имеется два прямых потомка - левый и правый. В левом потомке хранятся те адреса, значения которых меньше a, а в правом те адреса, значения которых больше b.

При обработке очередного адреса $x_j$ необходимо уметь вычислять его стековое расстояние. Его можно вычислить по формуле $dist(j) = j - P_j(x_j) - holes_j(P_j(x_j))$, где dist(j) - стековое расстояние адреса $x_j$; $P_j(z) = max\{i | 0 \leq i < j \ \& \ x_i = z\}$; $holes_j(i)$ - количество адресов, которых нет на стеке между моментами времени i и j. Для того, чтобы уметь вычислять значение функции holes, в каждом узле АВЛ-дерева вводится дополнительная ячейка памяти, в которой будет храниться количество адресов в правом поддереве.

На Рисунке 2 приведен пример извлечения адреса $x_{101}$ из стека (это равносильно добавлению числа 101 в АВЛ-дерево).

Как и в любом АВЛ-дереве, операции добавления и удаления вершин из дерева занимают $O(H)$ действий, где H - высота дерева. Вычисление значения функции $holes$ также занимает $O(H)$ операций. Отметим, что $H = O(logN)$. Таким образом,

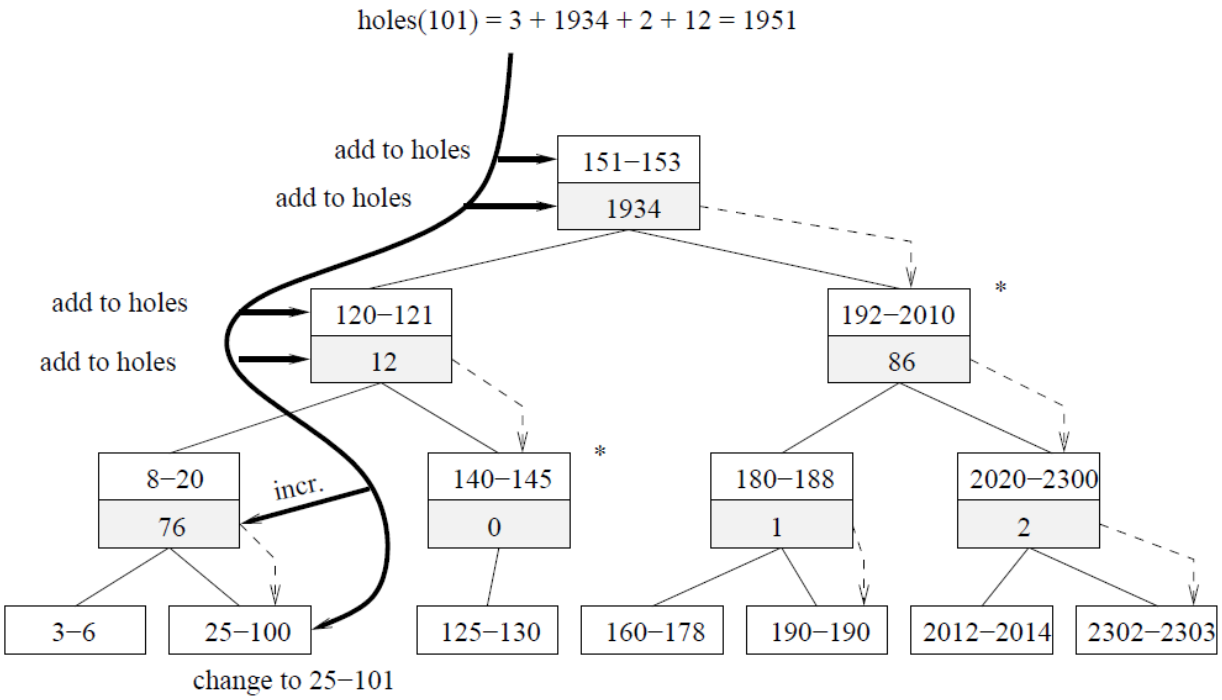


Рис. 2: Обновление интервального АВЛ-дерева

появляется возможность вычислять стековые расстояния и эмулировать работу стека LRU-алгоритма за логарифмическое время.

Преимущества интервального АВЛ-дерева по сравнению с другими сбалансированными деревьями:

1. В интервальном АВЛ-дереве в одном узле хранится несколько индексов. Это обстоятельство существенно уменьшает его высоту, позволяя эффективнее организовывать стандартные операции с АВЛ-деревом.

2. Многие операции добавления вершин в дерево проходят без его балансировки. Балансировка не требуется, например, в тех случаях, когда добавляемое в дерево число является соседним по отношению к некоторому интервалу(как, например, показано на Рисунке 2).

3. Сущесвенно вместо хранения адресов присутствующих в стеке, хранить те адреса, которые в стеке отсутствуют. Это вызвано тем, что адреса, однажды покинувшие стек, в него вернуться не могут. Таким образом, если элемент попал в АВЛ-дерево, то он гарантированно его никогда не покинет. Как следствие этого, требуется осуществлять меньшее количество балансировок дерева, что повышает производительность работы алгоритма.

# 3. Поток обращений к устройству хранения данных

Поток обращений - это последовательность элементарных запросов к системе хранения данных. Каждый запрос имеет четыре основные характеристики: время, адрес, размер и тип запроса. По сути каждая характерисика запроса является случайной величиной, зависящей от собственного набора параметров, которые могут меняться с течением времени. Можно считать, что каждая случайная величина независима от других, поэтому их можно анализировать независимо друг от друга.[1]

1. Эмпирически установлено, что случайная величина, равная разности времен между двумя последовательными запросами распределена экспоненциально.

2. Известно, что в большинстве практических ситуаций, стековые расстояния последовательности адресов распределены по закону Парето.

3. Размер запроса является дискретной случайной величиной, принимающий конечное множество значений, которое зависит от системы хранения. (Например, размер запросов в системе может принимать значения 0,5, 1, 2, 8, 64 КБайт). Для анализа будет достаточно рассматривать вероятности появления запроса данного размера.

4. Имеется только два типа запросов: чтение и запись. Тип запроса будет анализироваться по аналогии с пунктом 3, так как он также принимает конечное множество значений.

Обладая вышеуказанной информацией о характере потока обращений, появляется возможность разработать алгоритм управления кэш-памятью, учитывающий выявляемые статистические закономерности распределения потока обращений.

По последовательности запросов к устройству хранения нужно научиться восстанавливать параметры каждой из четырех случайных величин, входящих в состав потока запросов. После получения значений этих параметров, появляется возможность предсказывать поведение последующих запросов к системе хранения. Как следствие этого, появляется возможность эффективно управлять содержимым кэш-памяти, основываясь на собранной статистической информации.

Для наиболее эффективного управления кэш-памятью нужно наиболее точно восстанавливать параметры распределения каждой случайной величины. В следующих разделах приводятся описание и сравнение различных алгоритмов восстановления параметров распределений каждой из четырех случайных величин.

# 4. Алгоритмы восстановления параметров времен обращений

В данном разделе будут рассмотрены алгоритмы, анализирующие последовательность времен потока обращений. Пусть $\{t_i\}_{i=0}^n$ – последовательность времен обращений к устройству хранения. Получим новую последовательность $\{x_i\}_{i=1}^n$, такую что $x_i = t_i - t_{i-1}$, где $i = 1, 2 \dots n$. Известно, что $\{x_i\}_{i=1}^n$ была порождена экспоненциальным распределением с фиксированным параметром $\lambda$. По набору данных $\{x_i\}_{i=1}^n$ требуется получить наиболее правдоподобное значение параметра $\lambda$.

Существует множество методов восстановления параметра $\lambda$. Для экспоненциального распределения наиболее эффективным считается метод моментов (при присущей ему простоте реализации).[4] Его идея заключается в уравнивании математического ожидания экспоненциального распределения ($E_1$) и выборочного среднего последовательности $\{x_i\}_{i=1}^n (E_2)$.

$$E_1 = \frac{1}{\lambda}, E_2 = \frac{1}{n}\sum_{i=1}^{n} x_i, E_1 = E_2 \tag{1}$$

Решая данное уравнение, получим $\lambda = \frac{n}{\sum_{i=1}^n x_i}$. Данная оценка хорошо приближает значение параметра $\lambda$, однако не учитывает его вероятностное поведение. Например, для конкретной системы хранения может быть эмпирически установлено, что $\lambda \in [0, 1]$. При этом какие-то значения параметр $\lambda$ принимает чаще, а какие-то реже. Отсюда следует, что еще до начала анализа данных $\{x_i\}_{i=1}^n$, $\lambda$ уже обладает некоторым базовым распределением. Целесообразно придумать алгоритм, который бы, учитывая данное обстоятельство, вычислял бы $\lambda$ более точно, чем метод моментов.

Имеется некоторая случайная величина $\Lambda$, являющаяся параметром экспоненциального распределения Y. Таким образом, функция распределения Y зависит от случайной величины $\Lambda$.

$$F_Y(\Lambda, x) = (1 - e^{-\Lambda x})\chi_{[0;+\infty)}(x) = \begin{cases} 1 - e^{-\Lambda x} & , \quad x \geq 0 \\ 0 & , \quad x < 0 \end{cases} \tag{2}$$

Будем считать, что на вход программе подается набор вещественных чисел $x_1, ..., x_n$, каждое из которых является значением случайной величины Y в моменты времени $1, \dots, n$. Таким образом, имеется последовательность независимых случайных величин $X_1, \dots, X_n$, распределенных так же, как и Y. Положим $X = (X_1, \dots, X_n)$ - случайный n-мерный вектор.

Будем полагать, что $\Lambda$ является равномерно распределенной случайной величиной на некотором отрезке $[a, b]$, где $a > 0$.

$$p_\Lambda(x) = \frac{1}{b-a}\chi_{[a,b]}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases} \tag{3}$$

Имея информацию о распределении $\Lambda$, можно узнать функцию распределения вектора X, используя формулу полной вероятности.[3]

$$F_X(x) = F_{X_1,...,X_n}(x_1,...,x_n) = \int\limits_{-\infty}^{+\infty} \mathbf{P}(X_1 < x_1,...,X_n < x_n|\Lambda = t)p_\Lambda(t)dt =$$

$$= \frac{1}{b-a}\int\limits_a^b \prod_{i=1}^n P(X_i < x_i|\Lambda = t)dt = \frac{1}{b-a}\int\limits_a^b \prod_{i=1}^n (1 - \mathrm{e}^{-\mathrm{tx_i}})dt$$

Зная функцию распределения вектора X, можно вычислить его плотность:

$$p_X(x) = p_{X_1,...,X_n}(x_1,...,x_n) = \frac{\partial^n F_{X_1,...,X_n}(x_1,...,x_n)}{\partial x_1 ... \partial x_n} = \frac{1}{b-a}\int\limits_a^b t^n \prod_{i=1}^n \mathrm{e}^{-\mathrm{tx_i}}dt =$$

$$= \frac{1}{b-a}\int\limits_a^b t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}}dt$$

$$p_X(x) = \frac{1}{b-a}\int\limits_a^b t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}}dt \quad x \in \mathbf{R}^n \tag{4}$$

Найдем функцию совместного распределения случайных величин $X$ и $\Lambda$:

$$F_{X,\Lambda}(x) = F_{X_1,...,X_n,\Lambda}(x_1,...,x_n,\lambda) = P(X_1 < x_1,...,X_n < x_n, \Lambda < \lambda) =$$

$$= P(\Lambda < \lambda) \cdot P(X < x|\Lambda < \lambda) = F_\Lambda(\lambda)\chi_{[a,b]}(\lambda)\int\limits_{-\infty}^{\lambda} P(X_1 < x_1,...,X_n < x_n|\Lambda = t)p_\Lambda(t)dt =$$

$$= \frac{\lambda - a}{b-a}\chi_{[a,b]}(\lambda)\int\limits_a^\lambda \prod_{i=1}^n P(X_i < x_i|\Lambda = t) \cdot \frac{1}{b-a}dt = \frac{\lambda - a}{(b-a)^2}\chi_{[a,b]}(\lambda)\int\limits_a^\lambda \prod_{i=1}^n (1 - \mathrm{e}^{-\mathrm{tx_i}})dt$$

Вычислим плотность совместного распределения:

$$p_{X,\Lambda}(x,\lambda) = \frac{\partial^{n+1} F_{X_1,...,X_n,\Lambda}(x_1,...,x_n,\lambda)}{\partial x_1,...,\partial x_n,\partial\lambda} = \frac{\partial}{\partial\lambda}\left(\frac{\lambda - a}{(b-a)^2}\int\limits_a^\lambda t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}}dt\right)\chi_{[a,b]}(\lambda) =$$

$$= \left(\frac{1}{(b-a)^2}\int\limits_a^\lambda t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}}dt + \frac{\lambda - a}{(b-a)^2} \cdot (\lambda^n \mathrm{e}^{-\lambda\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}})\right)\chi_{[a,b]}(\lambda)$$

$$p_{X,\Lambda}(x,\lambda) = \left(\frac{1}{(b-a)^2}\int\limits_a^\lambda t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}}dt + \frac{\lambda - a}{(b-a)^2} \cdot (\lambda^n \mathrm{e}^{-\lambda\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}})\right)\chi_{[a,b]}(\lambda) \quad x \in \mathbf{R}^n, \lambda \in \mathbf{R} \tag{5}$$

Для каждого $n, x_1, \ldots, x_n$ можно получить плотность распределения случайной величины $\Lambda$ при условии $X_1 = x_1, \ldots, X_n = x_n$. Получим последовательность случайных

величин $\eta_n$. Их плотности можно вычислить по формуле $p_{\eta_n}(x,\lambda) := \frac{p_{X,\Lambda}(x,\lambda)}{p_X(x)}$

Далее, если считать, что $n, x_1, \ldots, x_n$ являются фиксированными числами, то получим случайную величину $\xi$, плотность которой зависит только от $\lambda$.

$p_\xi(\lambda) = p_{\eta_n}(x_1, \ldots, x_n, \lambda) \quad \forall \lambda \in \mathbf{R}$.

$$p_\xi(\lambda) = \left( \frac{\frac{1}{(b-a)^2} \int\limits_a^\lambda t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}} dt + \frac{\lambda - a}{(b-a)^2} \cdot (\lambda^n \mathrm{e}^{-\lambda \sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}})}{\frac{1}{b-a} \int\limits_a^b t^n \mathrm{e}^{-\mathrm{t}\sum\limits_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}} dt} \right) \chi_{[a,b]}(\lambda) \tag{6}$$

Искомым $\lambda$ нужно положить точку $\lambda^* \in (0, +\infty)$, в которой достигается максимум $p_\xi(\lambda)$. Найдем все точки $\lambda \in [a; b]$, в которых $p_\xi^{'}(\lambda) = 0$. Для краткости записи положим $c := \sum\limits_{i=1}^{n} x_i$.

$p_\xi^{'}(\lambda) = 0 \Leftrightarrow \frac{\partial}{\partial \lambda} \int\limits_a^\lambda t^n \mathrm{e}^{-\mathrm{ct}} dt + (\lambda - a) \cdot (\lambda^n \mathrm{e}^{-\mathrm{c}\lambda}) = 0 \Leftrightarrow \lambda^n e^{-c\lambda} + \lambda^n e^{-c\lambda} + (\lambda - a)(n\lambda^{n-1} e^{-c\lambda} + \lambda^n (-c) e^{-c\lambda}) = 0 \Leftrightarrow 2\lambda^n e^{-c\lambda} + n\lambda^n e^{-c\lambda} - c\lambda^{n+1} e^{-c\lambda} - an\lambda^{n-1} e^{-c\lambda} + ac\lambda^n e^{-c\lambda} = 0 \Leftrightarrow 2\lambda^n + n\lambda^n - c\lambda^{n+1} - an\lambda^{n-1} + ac\lambda^n = 0 \Leftrightarrow c\lambda^2 - (ac + n + 2)\lambda + an = 0$

$D = (ac + n + 2)^2 - 4acn = (ac - n + 2)^2 + 8n$

$$\lambda_{1,2} = \frac{ac + n + 2 \mp \sqrt{(ac - n + 2)^2 + 8n}}{2c} \tag{7}$$

Очевидно, что максимум достигается в точке $\lambda_2$. Докажем, что $0 < \lambda_1 < a$.

1. $\lambda_1 < a \Leftrightarrow ac + n + 2 - \sqrt{(ac - n + 2)^2 + 8n} < 2ac \Leftrightarrow (n + 2 - ac)^2 < (ac - n + 2)^2 + 8n \Leftrightarrow (n + 2 - ac - ac + n - 2)(n + 2 - ac + ac - n + 2) < 8n \Leftrightarrow n - ac < n \Leftrightarrow ac > 0$ - истинно.

2. $\lambda_1 > 0 \Leftrightarrow (ac + n + 2)^2 > (ac - n + 2)^2 + 8n \Leftrightarrow 2n(2ac + 4) > 8n \Leftrightarrow ac > 0$ - истинно.

Докажем, что $\lambda_2 > a$.

$\lambda_2 = \frac{ac + n + 2 + \sqrt{(ac - n + 2)^2 + 8n}}{2c} > \frac{ac + n + 2 + (ac - n + 2)}{2c} = \frac{2ac + 4}{2c} = a + \frac{4}{2c} > a$

В том случае, если $\lambda_2 > b$, то функция $p_\xi(\lambda)$ является монотонно возрастающей на всем отрезке $[a, b]$. В этом случае в качестве искомого значения $\lambda^*$ нужно принять $\lambda^* = b$. Получим окончательный результат:

$$\lambda^* = min \left\{ \frac{ac + n + 2 + \sqrt{(ac - n + 2)^2 + 8n}}{2c}, b \right\} \tag{8}$$

Однако, если мы не обладаем никакой информацией не только о распределении $\Lambda$, но и о ее области возможных значений, то метод перестает быть применимым. Отсюда возникает мысль о совмещении двух методов воедино.

Предлагается в качестве базового приближения значения $\lambda$ взять выражение $\lambda^* = \frac{n}{\sum\limits_{i=1}^{n} x_i}$. Далее в зависимости от входных данных $\{x_i\}_{i=1}^n$ выбрать значения a и b так,

чтобы $\lambda^* \in [a, b]$ и вероятность того, что истинное значение $\lambda \in [a, b]$ было бы очень велико. После этого для выбранных значений a и b можно применить второй алгоритм. Полученное значение будет приближать оптимальное значение $\lambda$ лучше, чем $\lambda^*$.

Для применения второго алгоритма необходимо обладать статистической информацией о базовом распределении параметра $\lambda$. Очевидно, что для различных систем хранения базовые распределения будут различаться.

Каждый алгоритм тестировался на одних и тех же входных данных. Было получено 1 000 файлов, каждый из которых состоял из 1 000 чисел, сгенерированных по экспоненциальному распределению с известным параметром $\lambda_i$, где $i = 1, \ldots, 1000$. К данным каждого файла применялся алгоритм и, таким образом, было получено 1 000 приближений исходных параметров $\lambda_i^*$. Зная истинные значения параметров, с помощью которых генерировались тестовые множества, можно получить относительную погрешность по формуле $k_i = \frac{\lambda_i - \lambda_i^*}{\lambda_i}$ . Далее вычислялись математические ожидания и дисперсии относительной погрешности. Результаты применения алгоритмов приведены в Таблицах 1, 2. В таблицах приведены матожидания и дисперсии относительной погрешности при параметрах $a = 0.1, 0.5, 0.9, 0.95$ и фиксированном параметре $b = 1.0$. Для алгоритма 3 выбирался отрезок $[\lambda^* - 0.2, \lambda^* + 0.1]$, где $\lambda^*$ вычислялся по методу моментов.

Таблица 1: Матожидание относительной погрешности.

| | a = 0.1 | a = 0.5 | a = 0.9 | a = 0.95 |
|---|---|---|---|---|
| Алгоритм 1 | 0.0264 | 0.0254437 | 0.0264689 | 0.025155 |
| Алгоритм 2 | 0.026965 | 0.02673426 | 0.028047 | 0.023228 |
| Алгоритм 3 | 0.27016 | 0.02627 | 0.02333 | 0.018956 |

Таблица 2: Дисперсия относительной погрешности

| | a = 0.1 | a = 0.5 | a = 0.9 | a = 0.95 |
|---|---|---|---|---|
| Алгоритм 1 | $4.06 \cdot 10^{-4}$ | $3.69 \cdot 10^{-4}$ | $3,63 \cdot 10^{-4}$ | $3.64 \cdot 10^{-4}$ |
| Алгоритм 2 | $4.22 \cdot 10^{-4}$ | $3.94 \cdot 10^{-4}$ | $3.88 \cdot 10^{-4}$ | $1.90 \cdot 10^{-4}$ |
| Алгоритм 3 | $4.42 \cdot 10^{-4}$ | $3.96 \cdot 10^{-4}$ | $3.068 \cdot 10^{-4}$ | $1.99 \cdot 10^{-4}$ |

Из представленных в таблицах данных можно сделать вывод, что при уменьшении длины отрезка $[a, b]$ алгоритмы 2 и 3 становится более точными, чем алгоритм 1. Однако при достаточно большой длине отрезка $[a, b]$ самым точным методом восстановления параметра $\lambda$ является метод моментов.

# 5. Алгоритмы восстановления параметров потока адресов

По последовательности адресов при помощи интервального АВЛ-дерева эмулировалась работа LRU-алгоритма, в результате чего была получена последовательность стековых расстояний, которая, как говорилось ранее, распределена по закону Парето.

$$p(x) = \begin{cases} \frac{kx_m^k}{x^{k+1}}, & x \in [x_m, +\infty) \\ 0, & x \in (-\infty, x_m) \end{cases}$$

$$F(x) = \begin{cases} 1 - (\frac{x_m}{x})^k, & x \in [x_m, +\infty) \\ 0, & x \in (-\infty, x_m) \end{cases}$$

Для восстановления параметров распределения Парето можно воспользоваться методом моментов. Нетрудно убедиться в том, что матожидание $EX = \frac{kx_m}{k-1}$, а второй момент $EX^2 = \frac{kx_m^2}{k-1}$.

Пусть $\{x_i\}_{i=1}^n$ - последовательность стековых расстояний, распределенных по закону Парето. Для этой последовательности возможно за линейное время вычислить первый и второй выборочные моменты ($E_1$ и $E_2$).

$$E_1 = \frac{1}{n}\sum_{i=1}^{n} x_i \quad E_2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2$$

Решим систему уравнений:

$$\begin{cases} \frac{kx_m}{k-1} & = E_1 \\ \frac{kx_m^2}{k-1} & = E_2 \end{cases}$$

Подставив первое уравнение во второе, получим $E_1 x_m = E_2 \Leftrightarrow x_m = E_2/E_1$. Из первого уравнения получим

$$\frac{k}{k-1} = E_1 : \frac{E_2}{E_1} = \frac{{E_1}^2}{E_2}$$

Отсюда следует, что

$$\frac{k}{k-1} = \frac{{E_1}^2}{E_2} \Leftrightarrow kE_2 = k{E_1}^2 - {E_1}^2 \Leftrightarrow k = \frac{{E_1}^2}{{E_1}^2 - E_2} \Leftrightarrow k = 1 + \frac{E_2}{{E_1}^2 - E_2}$$

Заметим, что второй выборочный момент $E_2$ всегда неотрицателен. Заметим также, что выражение ${E_1}^2 - E_2$ является выборочной дисперсией последовательности

$\{x_i\}_{i=1}^{n}$. Получим окончательный результат:

$$\begin{cases} x_m & = \frac{E_2}{E_1} \\ k & = \frac{E_2}{E_2 - {E_1}^2} \end{cases}$$

Параметр $x_m$ всегда должен получаться меньше 1,5. Докажем это утверждение от противного. Пусть $x_m > 1,5$. Тогда $P(X \leq 1,5) \leq P(X < x_m) = 0 \Rightarrow P(X \leq 1,5) = 0, \Rightarrow X > 1,5$ что в свою очередь означает, что в результате округления случайной величины X к ближайшему целому результат всегда будет не меньше 2. Но в этом случае, стековые расстояния не могут принимать значения равные единице, что приводит нас к противоречию с предположением, что $x_m > 1,5$.

Использование идей, примененных к выводу формулы (6), не приводит к успеху, поскольку в процессе нахождения экстремумов условной плотности вероятности мы сталкиваемся с проблемой длительности их расчетов в вычислительных системах, что снижает производителность систем в целом. Поэтому для восстановления параметров последовательности адресов целесообразно применять описанный выше метод моментов.

# 6. Анализ размера и типа запросов потока обращений

Размер и тип запроса, как упомяналось ранее, являются дискретными случайными величинами, принимающими конечное множество значений.

Пусть $\{s_i\}_{i=1}^n$ - последовательность размеров запросов к устройству хранения данных. Пусть случайная величина S принимает множество значений $\{v_i\}_{i=1}^m$. Тогда вероятность того, что случайная величина S принимает значение $v_i$, можно приближенно принять равной частоте появления $v_i$ в последовательности $\{s_i\}_{i=1}^n$.

$$\mathbf{P}(S = v_i) = \frac{|\{j : s_j = v_i\}|}{n} \quad \forall i = 1, \dots m \tag{9}$$

Рассуждая подобным образом, можно получить аналогичные формулы для типа запроса T (чтение и запись).

$$\mathbf{P}(T = READ) = \frac{|\{j : t_j = READ\}|}{n}$$

$$\mathbf{P}(T = WRITE) = \frac{|\{j : t_j = WRITE\}|}{n}$$

Приведенные выше формулы описывают распределения случайных величин S и T.

# Заключение

В работе был приведен алгоритм быcрого восстанавления стековых расстояний по- следовательности адресов. Были описаны основные характеристики потока обращений к устройству хранения, а также были указаны эмпирически выявленные закономер- ности распределения потока обращений. Получены алгоритмы восстановления параметров потока обращений при условии их постоянства во времени, а также проведен их сравнительный анализ. Была разработана программа (GUI - C#, вычисления - C++), применяющая каждый из полученных алгоритмов к потоку обращений. Результаты ее исполнения подтверждают применимость вышеописанных методов восстановления параметров потока обращений.

# Список литературы

[1] Feitelson Dror G. Workload Modeling for Computer Systems Performance Evaluation. — The Hebrew University of Jerusalem.

[2] George Alm´asi C˘alin Cascaval David A. Padua. Calculating Stack Distances Efficiently. — University of Illinois at UrbanaChampaign.

[3] А.Н.Бородин. Элементарный курс теории вероятностей и математической статистики.

[4] Г.Б.Ходасевич. Обработка экспериментальных данных на ЭВМ.