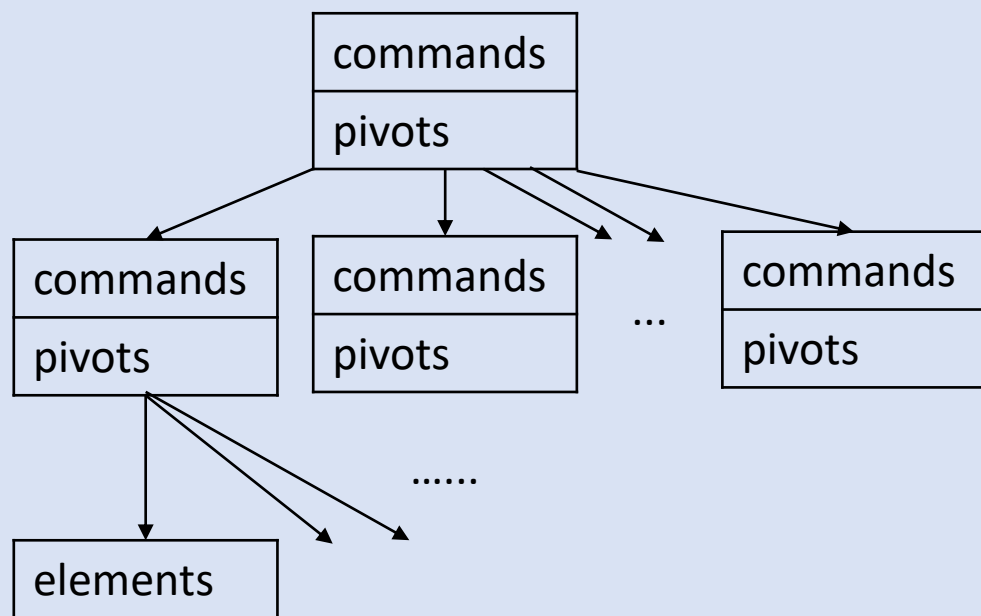


Основы построения файловых систем



B^ϵ -деревья



B^ϵ -деревья объединяют В-дерево и журнал.

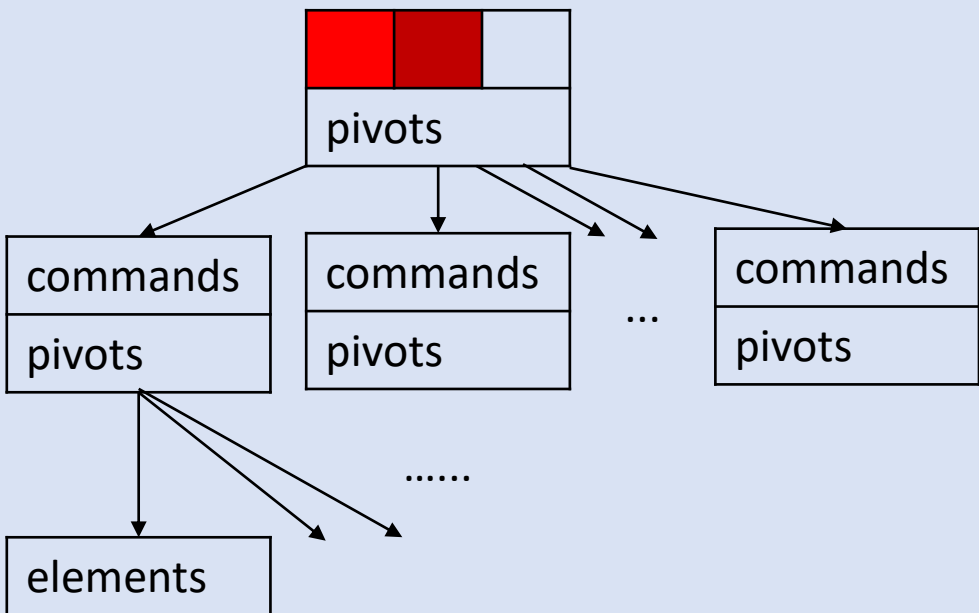
Как и в В-деревьях, узлы являются достаточно длинными блоками, но теперь они разделяются на две части:

- pivots (пары ключ-значение с указателями на пользовательские данные или на другие узлы дерева),
- commands (журнал вставок и удалений).

Узлы длиной B байт делятся в пропорции B^ϵ байт на pivots и $B - B^\epsilon$ байт на commands.

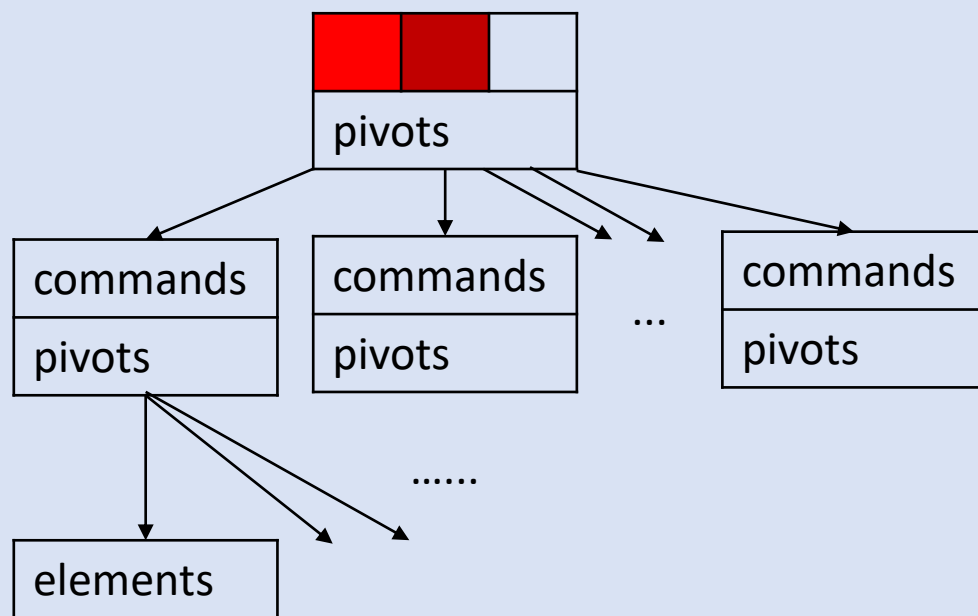
https://www.usenix.org/system/files/login/articles/login_oct15_05_bender.pdf

B^{ϵ} -деревья



- Вставки и удаления добавляются только в журнал корневого узла,
- При переполнении журнала в корне он выталкивается в журналы дочерних узлов, причём выталкиваются только модификации наиболее изменённых поддеревьев.
- В частности, число изменений, которые выталкиваются в поддерево, не меньше $(B - B^\epsilon) / B^\epsilon \approx B^{1-\epsilon}$. Таким образом, амортизированное время вставки составляет $O\left(\frac{\log_B N}{B^\epsilon}\right)$.

B^{ϵ} -деревья



У журнала есть ещё преимущества: записи в нём можно трактовать не как добавления элементов, а как команды, произвольно модифицирующие поддеревья.

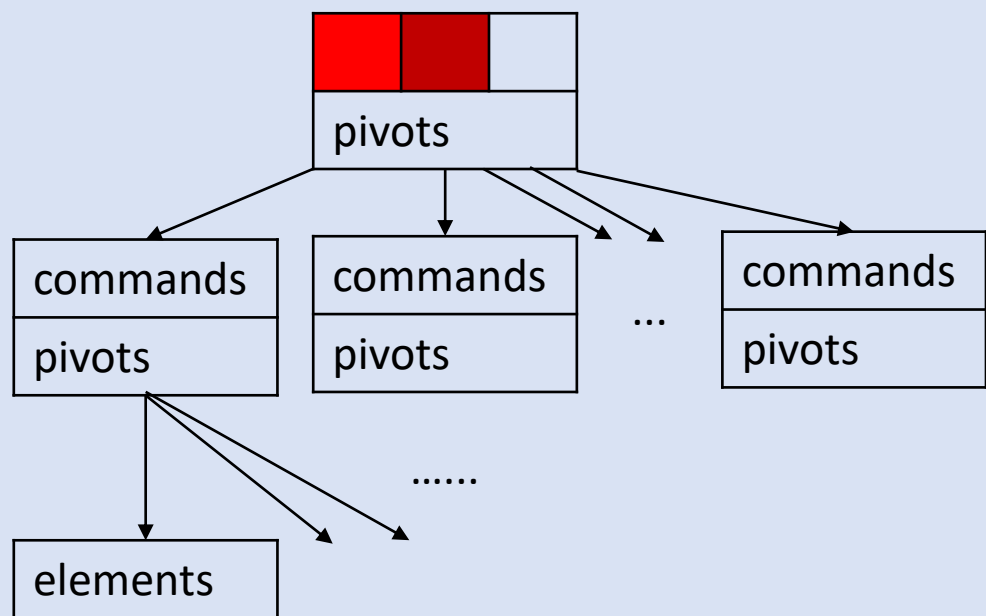
Примеры:

- update,
- upsert (UPdate or inSERT),
- range delete.

Как следствие, для B^{ϵ} -деревьев возможна эффективная реализация операции “range query” (получить все элементы множества, попадающие в указанный диапазон).

Какая сложность этой операции у B^{ϵ} -дерева и у LSM-дерева?

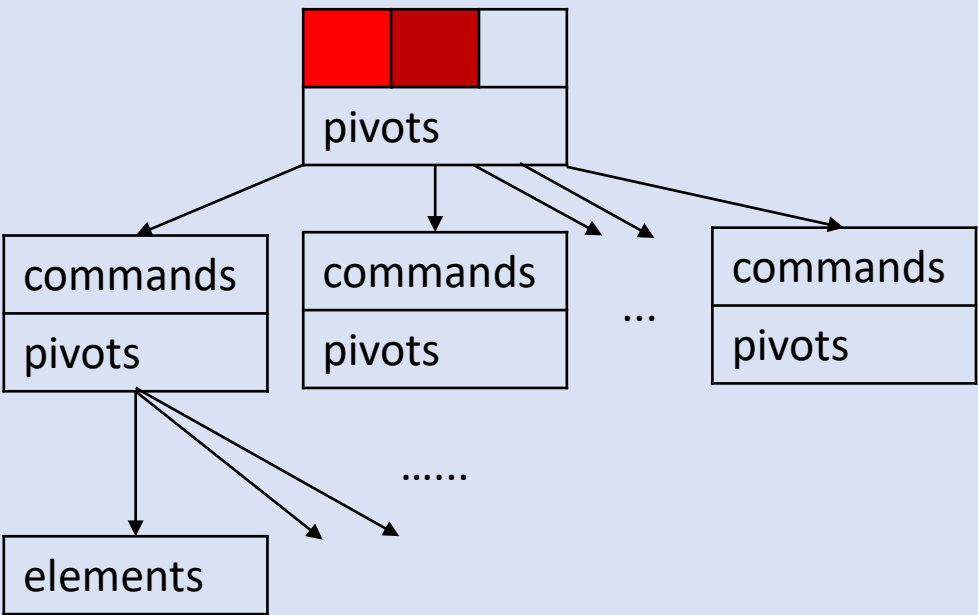
B^ϵ -деревья



Преимущества такой реализации:

- Нет необходимости поиска во многих деревьях или построения фильтров Блума,
- Журнал изменений расположен в корневом узле, в котором происходит большая часть изменений, всегда в кеше,
- При расщеплении журнала изменений генерируется меньше IO, чем при слиянии компонент LSM-дерева,
- Размер узлов B^ϵ -деревьев можно делать много больше, чем у B-деревьев, что уменьшает их глубину.

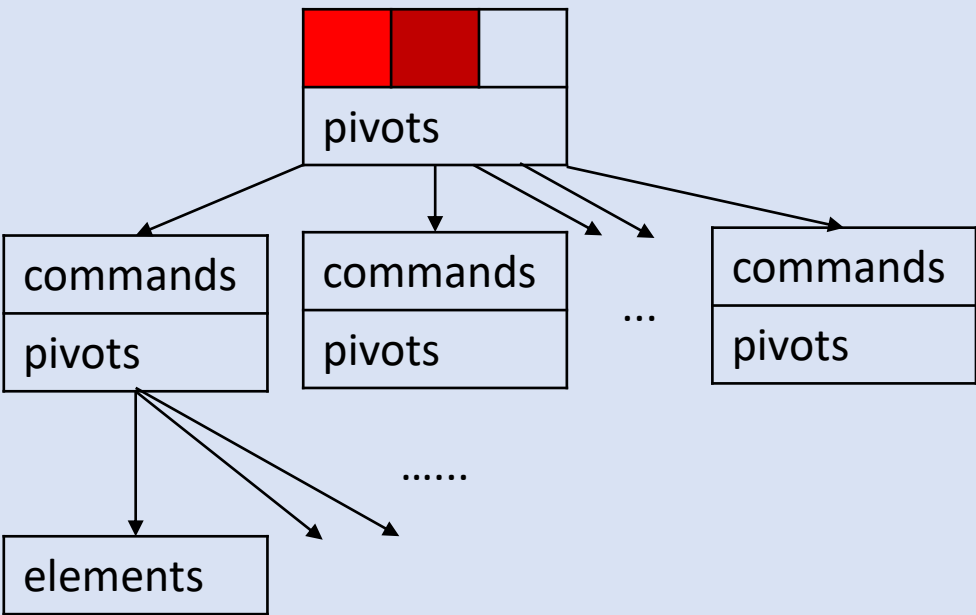
B^{ϵ} -деревья: журналирование и tree checkpointing



Файловые системы вроде ext4 и XFS ведут журнал физических изменений. Это много проще журнала логических изменений ФС, поскольку все операции в нём идемпотентны.

Что пишется в журнал B^{ϵ} -дерева: логические или физические изменения?

B^{ϵ} -деревья: журналирование и tree checkpointing

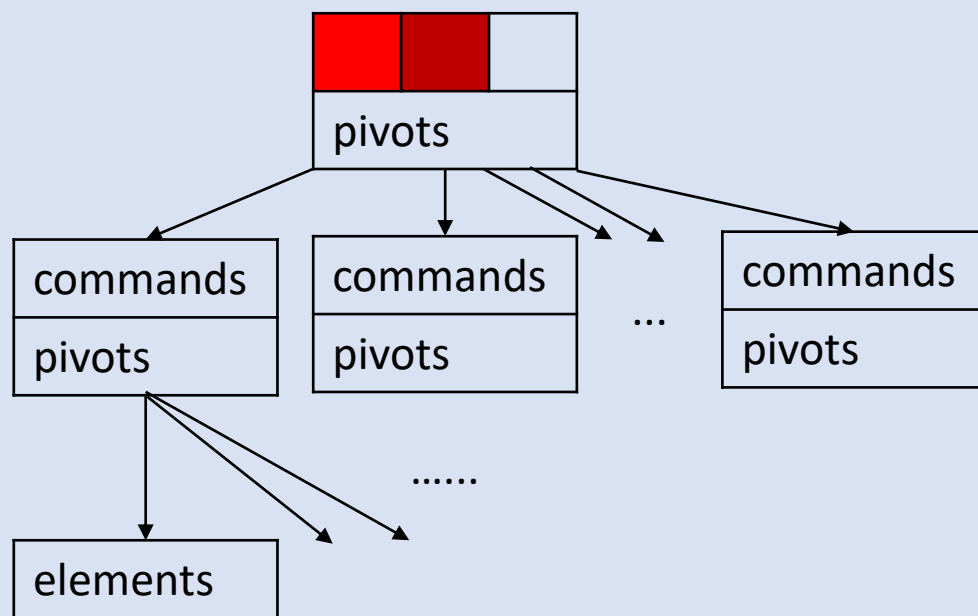


Файловые системы вроде ext4 и XFS ведут журнал физических изменений. Это много проще журнала логических изменений ФС, поскольку все операции в нём идемпотентны.

А в B^{ϵ} -дереве журналируются, наоборот, логические изменения.

В чём разница? Как делать crash recovery для B^{ϵ} -дерева и не получить проблем с неидемпотентностью?

B^{ϵ} -деревья: журналирование и tree checkpointing



Файловые системы вроде ext4 и XFS ведут журнал физических изменений. Это много проще журнала логических изменений ФС, поскольку все операции в нём идемпотентны.

А в B^{ϵ} -дереве журналируются, наоборот, логические изменения.

В чём разница? Как делать crash recovery для B^{ϵ} -дерева и не получить проблем с неидемпотентностью?

- Ссылка на корневой узел хранится в суперблоке дерева.
- Crash recovery не меняет узлы дерева, а только выписывает изменённые.
- Когда изменения полностью выписаны, обновляется указатель на корень в суперблоке.
- Если crash recovery выполняется второй раз, то она начинается со старого суперблока, т.е. с того же дерева, что и прерванная, поэтому никакая запись в журнале не применяется дважды.

Напоминание: B-деревья, LSM-деревья, B^{ϵ} -деревья

- Бинарные деревья поиска.
Каждый узел содержит один ключ и имеет только двух потомков
 - - имеют большую глубину,
 - - генерируют много случайного ввода-вывода.

Напоминание: В-деревья, LSM-деревья, B^{ϵ} -деревья

- Бинарные деревья поиска.

Каждый узел содержит один ключ и имеет только двух потомков

- - имеют большую глубину,
- - генерируют много случайного ввода-вывода.

- В-деревья.

Каждый узел – блок на диске. В нём умещается несколько пар ключ-значение.

- + меньше глубина по сравнению с бинарным деревом,
- + чтение одного узла линейное и даёт сразу много пар ключ-значение,
- - узлы дерева заполняются не полностью,
- - удаление создаёт мусор и уменьшает долю полезного занятого места.

Напоминание: B-деревья, LSM-деревья, B^{ϵ} -деревья

- B-деревья.

Каждый узел – блок на диске. В нём умещается несколько пар ключ-значение.

- + меньше глубина по сравнению с бинарным деревом,
- + чтение одного узла линейное и даёт сразу много пар ключ-значение,
- - узлы дерева заполняются не полностью,
- - удаление создаёт мусор и уменьшает долю полезного занятого места.

- LSM-деревья.

Иерархия полностью заполненных B-деревьев.

- + нет незанятого места в узлах,
- + как следствие, у узлов больше потомков (больше fan-out factor),
- + большее число потомков у узлов влечёт меньшую глубину дерева,
- - поиск требует поиска в нескольких деревьях или применения фильтра Блума,
- - сильно неравномерное распределение времен вставок элементов,
- - write amplification: вставка элемента может привести к переписыванию многих деревьев.

Напоминание: B-деревья, LSM-деревья, B^{ϵ} -деревья

- LSM-деревья.

Иерархия полностью заполненных B-деревьев.

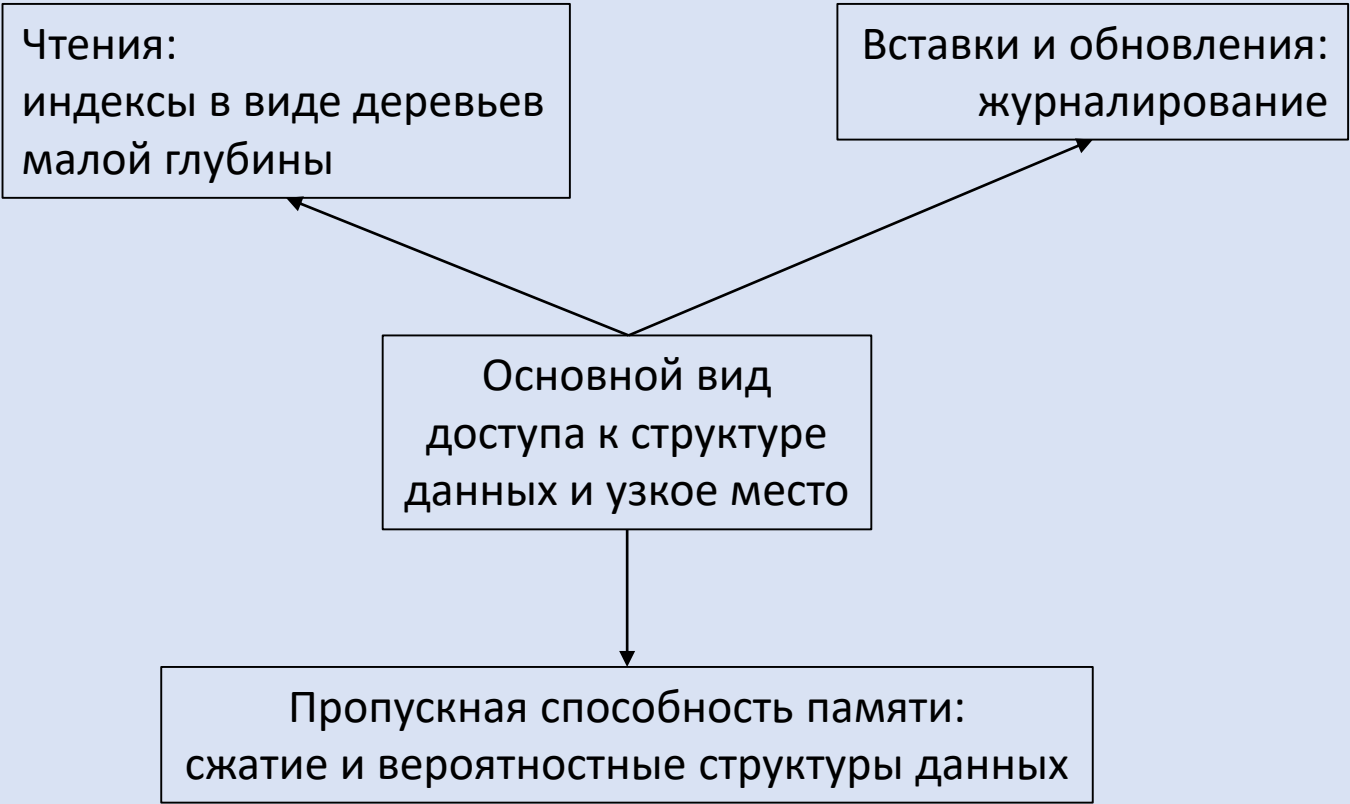
- + нет незанятого места в узлах,
- + как следствие, у узлов больше потомков (больше fan-out factor),
- + большее число потомков у узлов влечёт меньшую глубину дерева,
- - поиск требует поиска в нескольких деревьях или применения фильтра Блума,
- - сильно неравномерное распределение времен вставок элементов,
- - write amplification: вставка элемента может привести к переписыванию многих деревьев.

- B^{ϵ} -деревья.

Одно B-дерево, где в каждом узле, помимо пар ключ-значение, есть журнал изменений в поддереве.

- + вставки делаются только в журнал,
- + запись в журнал на диске линейная и делается только в один блок,
- + при переполнении журнала части его выталкиваются в журналы только наиболее изменённых потомков, поэтому переписывается сильно меньшая часть дерева, чем в случае B-tree или LSM-tree,
- + поддерживает range queries – запросы на получение значений, соответствующих ключам из диапазона.

Read, write, and space amplification



В рассмотренных сегодня конструкциях получалось улучшить один или два параметра за счёт остальных:

- В-дерево имеет лучшее время поиска, чем rb-дерево (если говорить о хранении на диске, а не в памяти), но занимает больше места на диске из-за не полностью заполненных узлов,
- LSM-дерево обеспечивает гораздо более быстрые вставки в типичном случае, чем В-дерево, но асимптотика поиска в нём хуже,
- Чтобы исправить асимптотику поиска в LSM-дереве, мы добавили фильтры Блума и потеряли немного места и ресурсов CPU,
- Сжатие данных в дереве может существенно уменьшать его размер на диске за счёт времени вставки и поиска,
- Увеличение размеров узлов В-дерева улучшает время поиска за счёт большего использования места и больших затрат при вставках.

Напоминание: Bloom filters

Поиск в LSM-дереве приходится реализовывать как несколько поисков по его составляющим разных уровней.

Можно избежать поиска во многих деревьях T_i , если научиться быстро определять, что искомого ключа в T_i не содержится. Это делает фильтр Блума, вероятностная структура данных, которая по множеству и ключу может выдавать ответы

- элемента в множестве нет,
- элемент в множестве может присутствовать.

Конструкция фильтра Блума: пусть имеется битовый массив длиной m и k независимых хеш-функций f_i , принимающих значения в диапазоне $[0, m-1)$.

- При вставке элемента x установим в 1 биты, стоящие на местах $f_1(x), f_2(x), \dots, f_k(x)$,
- Для проверки отсутствия элемента y проверим, установлены ли биты на позициях $f_1(y), f_2(y), \dots, f_k(y)$.

Фильтр Блума полезен в ситуациях, когда надо быстро определить, что элемент в множество не входит, чтобы избежать дорогостоящего поиска по множеству.

Power of two choices

Зачем в фильтре Блума использовать несколько независимых хешей?

Power of two choices

Зачем в фильтре Блума использовать несколько независимых хешей?

Факт 1: Пусть дана хеш-таблица, где элементы размещаются по хешу в N списков. Вставим в неё N случайных элементов. Какова будет длина максимально заполненного списка? С вероятностью $\geq 1 - O(1/N)$ она будет равна

$$\frac{\log N}{\log \log N} + O(1)$$

Power of two choices

Зачем в фильтре Блума использовать несколько независимых хешей?

Факт 1: Пусть дана хеш-таблица, где элементы размещаются по хешу в N списков. Вставим в неё N случайных элементов. Какова будет длина максимально заполненного списка? С вероятностью $\geq 1 - O(1/N)$ она будет равна

$$\frac{\log N}{\log \log N} + O(1)$$

Факт 2: Пусть дана хеш-таблица, где элементы размещаются в N списков, но правило размещения таково: при вставке элемента посчитаем для него d независимых хешей и выберем самый короткий список, соответствующий одному из полученных хешей.

Вставим N случайных элементов в такую хеш-таблицу. Какова будет длина максимально заполненного списка на этот раз? С вероятностью $\geq 1 - O(1/N)$ она составит

$$\frac{\log \log N}{\log d} + O(1)$$

Использование двух хеш-функций вместо одной улучшает асимптотику длины наиболее занятого списка. Использование большего числа только уменьшает константу в $O()$.

Power of two choices

Применения:

- хеш-таблицы,
- фильтры Блума,
- ?

Power of two choices

Применения:

- хеш-таблицы,
- фильтры Блума,
- балансировка нагрузки в распределённых системах,
- уменьшение tail latency в распределённых системах.

Power of two choices

Применения:

- хеш-таблицы,
- фильтры Блума,
- балансировка нагрузки в распределённых системах,
- уменьшение tail latency в распределённых системах.

Пример: если есть несколько HTTP-серверов, с которых можно скачать файл, то можно очень просто распределить нагрузку между ними:

1. выбрать два случайных сервера,
2. отправить обоим один и тот же запрос,
3. с того, кто первым начнёт слать ответ, скачать файл,
4. у более медленного отменить запрос.

Проблема: удвоение числа запросов (не считая запросов на отмену).

https://people.cs.umass.edu/~ramesh/Site/PUBLICATIONS_files/MRS01.pdf

Power of two choices

Применения:

- хеш-таблицы,
- фильтры Блума,
- балансировка нагрузки в распределённых системах,
- уменьшение tail latency в распределённых системах.

Вопрос: пусть у нас есть N одинаковых серверов, которые 99% запросов обрабатывают $< 10\text{ms}$, а 1% запросов (случайных) обрабатывают 1s . Если для построения ответа пользователю требуется получить ответ от 10 серверов, то какая доля пользовательских запросов будет обработана за 10ms ?

<http://cseweb.ucsd.edu/~gmporter/classes/fa17/cse124/post/schedule/p74-dean.pdf>

Power of two choices

Применения:

- хеш-таблицы,
- фильтры Блума,
- балансировка нагрузки в распределённых системах,
- уменьшение tail latency в распределённых системах.

Идея: послать запрос одному серверу и, если время ответа превысило 90-й (95-й) персентиль, то перепослать запрос уже другому серверу.

Заодно это решает проблему удвоением числа запросов при наивном применении power of two choices.

<http://cseweb.ucsd.edu/~gmporter/classes/fa17/cse124/post/schedule/p74-dean.pdf>