Основы построения файловых систем

Cyclic Redundancy Check

Рассмотрим сообщение как последовательность битов (элеметов GF(2)) и сопоставим ему многочлен из $GF(2)[X]$:

$$a_{n-1}a_{n-2}\dots a_0 \rightsquigarrow M(X) = X^n + a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \dots + a_0$$

Возьмём многочлен $C \in GF(2)[X]$ степени m, посчитаем r(X) – остаток от деления $M(X) * X^m$ на $C(X)$.

Теперь построим многочлен

$$M(X) * X^m + r(X)$$

В чём отличие от построения кода Рида-Соломона?

Cyclic Redundancy Check

Рассмотрим сообщение как последовательность битов (элеметов GF(2)) и сопоставим ему многочлен из $GF(2)[X]$:

$$a_{n-1}a_{n-2}\dots a_0 \rightsquigarrow M(X) = X^n + a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \dots + a_0$$

Возьмём многочлен $C \in GF(2)[X]$ степени m, посчитаем r(X) – остаток от деления $M(X) * X^m$ на $C(X)$.
Теперь построим многочлен

$$M(X) * X^m + r(X)$$

В чём отличие от построения кода Рида-Соломона?

- Другая цель: проверить, что сообщение при передаче не было изменено,
- Более простые коэффициенты (GF(2) вместо GF(256)), что упрощает аппаратную реализацию (из арифметических операций остаётся только XOR).

Многочлен C(X) подбирается так, чтобы обеспечить обнаружение определённых типов ошибок.

Ошибки, которые находит CRC (упражнения)

- Если C(X) имеет два и более ненулевых коэффициентов, то он определяет изменение любого одного бита.
- Если в разложении C(X) на неприводимые множители есть многочлен степени m, то C(X) определяет изменение любых двух бит, расположенных на расстоянии, меньшем m.
- Если C(X) делится на X+1, то он определяет любую ошибку, меняющую нечётное число бит.

Примеры CRC

Название	Где применяется	Порождающий многочлен*
CRC-16-CCITT	Bluetooth	0x1021
CRC-16-IBM	USB	0x8005
CRC-32	Ethernet, SATA, MPEG-2, gzip, bzip2, PNG	0x04C11DB7
CRC-32C (Castagnoli)	iSCSI, SCTP, SSE4.2, btrfs, ext4, Ceph	0x1EDC6F41

** Отдельные биты числа рассматриваются как коэффициенты порождающего многочлена.*

Криптографические хеши

CRC очень просты в вычислении и обнаруживают простые ошибки. Но их легко обмануть намеренными ошибками.

Для надёжной проверки того, что блок данных не был повреждён или изменён, применяются криптографические хеши:

- MD4, MD5
- SHA1, SHA-256, SHA-384, SHA-512.

На них полагаются, поскольку сейчас не известно алгоритмов поиска коллизий этих хешей, кроме перебора:

- Для MD4, MD5 и SHA1 – большого числа вариантов,
- Для SHA-256 и старше – полного перебора.

Примерная скорость вычисления хешей и CRC*

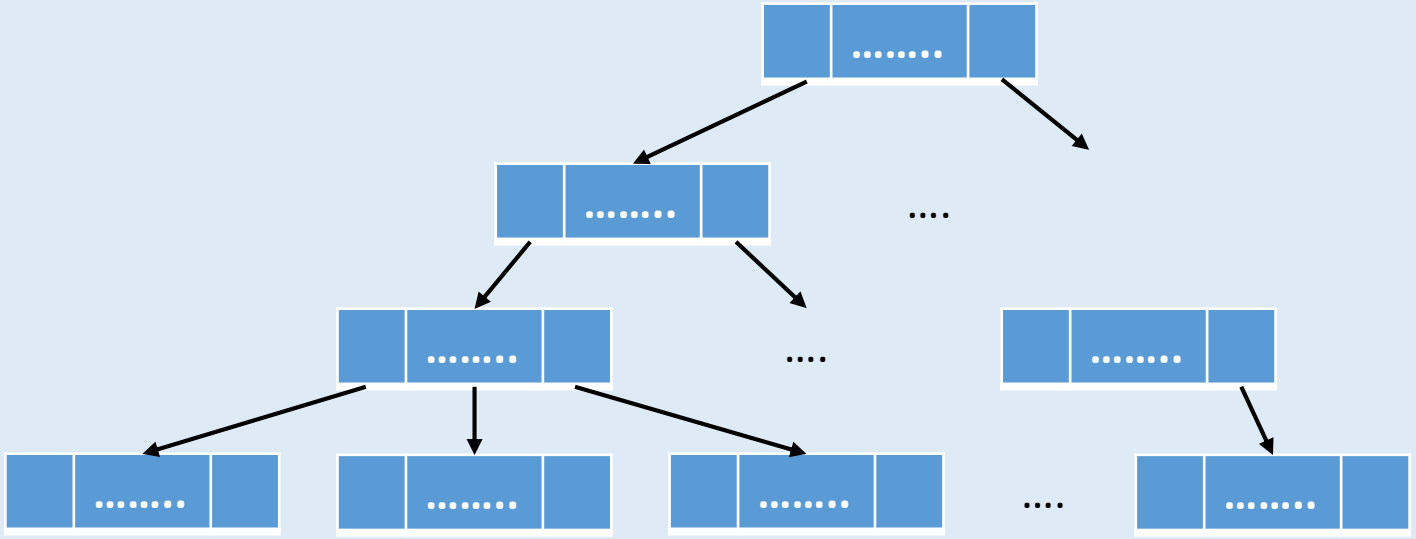
Алгоритм	MB/sec	Cycles/byte
CRC32	253	6.9
Adler32**	920	1.9
MD5	255	6.8
SHA-1	153	11.4
SHA-512	99	17.7

SSE4.2 содержит инструкции для вычисления SHA. Одно ядро Haswell 2.2GHz считает SHA-512 со скоростью 300MB/sec. Оно же научилось быстрее считать CRC32: примерно 2.5GB/sec.

** Измерено на Core 2 1.83GHz*

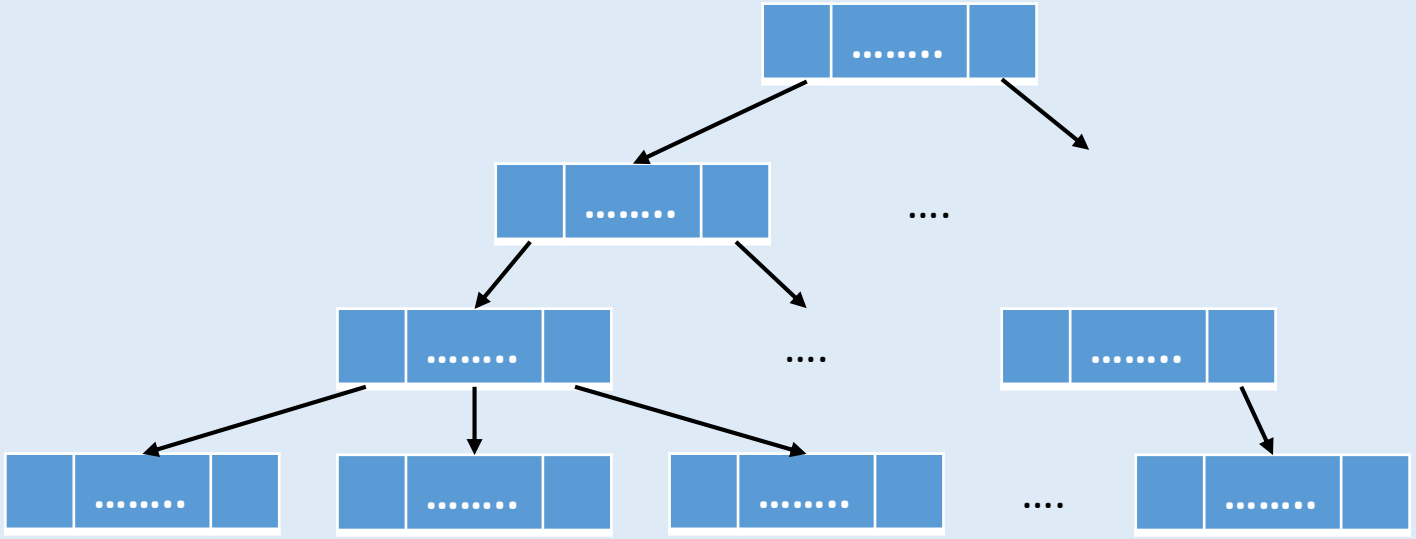
*** Не является CRC; используется в zlib*

B-деревья



- В узлах хранятся массивы пар (ключ, ссылка),
- Массивы имеют ограниченную длину:
  от k до 2k элементов,
- Указатели на страницы данных хранятся в листьях,
  во внутренних узлах – ссылки на страницы
  с узлами-потомками,
- Все листья расположены на одной глубине,
- Пара (k_i, p_i) ссылается на поддерево, все ключи которого находятся в диапазоне [k_i, k_{i+1}),
- Если при вставке происходит переполнение узла, то он разделяется на два узла длины k, а средний элемент перемещается в родительский узел.

B-деревья

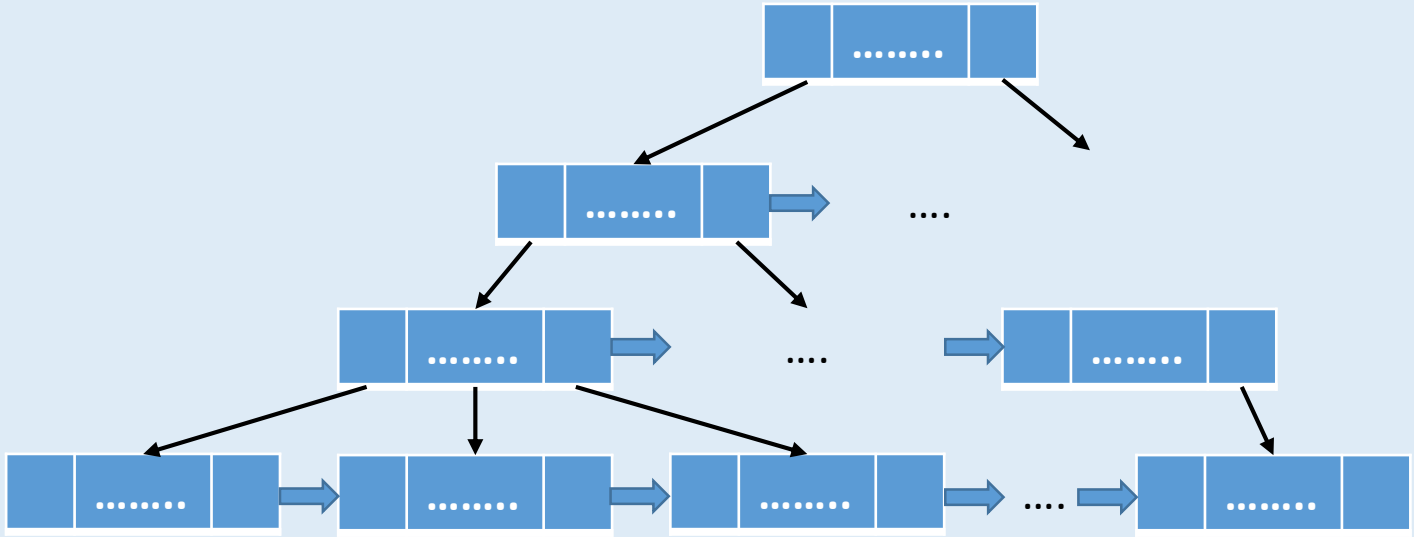


- В узлах хранятся массивы пар (ключ, ссылка),
- Массивы имеют ограниченную длину:
  от k до 2k элементов,
- Указатели на страницы данных хранятся в листьях,
  во внутренних узлах – ссылки на страницы
  с узлами-потомками,
- Все листья расположены на одной глубине,
- Пара (k_i, p_i) ссылается на поддерево, все ключи которого находятся в диапазоне [k_i, k_{i+1}),
- Если при вставке происходит переполнение узла, то он разделяется на два узла длины k, а средний элемент перемещается в родительский узел.

Есть проблемы:

- Вставки и удаления создают случайное IO,
- Удаление зачастую реализуется нетривиально, или оставляет много мусора,
- В многопоточной среде надо брать блокировки сразу на весь путь до листа.

B^{link}-деревья (Lehman, Yao)*



При расщеплении узла не обязательно модифицировать родителя – хватит проставить ссылку на правого соседа, а родительский узел можно модифицировать потом.

В итоге, в каждый момент времени достаточно держать блокировку только на одном узле.

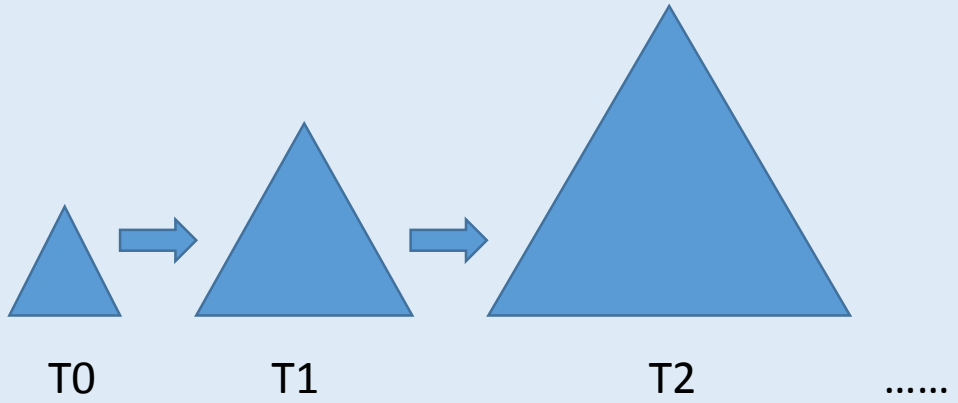
https://www.csd.uoc.gr/~hy460/pdf/p650-lehman.pdf

** Используются, например, в PostgreSQL.*

Log-Structured Merge Tree

LSM-дерево – это иерархия B-деревьев.

- Дерево T_0 (возможно, несколько первых) располагается в RAM, гарантируя быструю вставку.
- При переполнении дерева T_i оно сливается с деревом T_{i+1}; полученное дерево объявляется новым T_{i+1}.
- Поиск элемента делается по очереди в деревьях T_0, T_1, …
- Удаление элемента реализуется как вставка элемента, помеченного флагом «удалённый». Фактическое удаление произойдёт при слиянии деревьев.



Преимущества такого подхода:

- Деревья в RAM гарантируют очень быстрые вставки и удаления,
- Процедура слияния деревьев генерирует последовательное IO.

Факт: оптимальная производительность вставок в LSM-дерево (наименьшие накладные расходы на слияния) достигается, если число элементов в деревьях T_0, T_1, … образует геометрическую прогрессию.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.2782&rep=rep1&type=pdf
http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

Домашнее задание

1. Написать B-дерево, которое хранит пары из 64-битного ключа и 64-битного значения. Удаление сделать как вставку маркера удаления.
2. Написать функцию для слияния двух B-деревьев.
3. Померять скорость вставки элементов со случайными ключами.
4. Написать программу, которая обходит домашний каталог, режет файлы на куски по 1Kb, и от этих кусков считает
   - CRC32 (можно взять в zlib),
   - Fletcher32 или Adler32,
   - MD5 (см. документацию на openssl),
   - SHA-256.

   Построить гистограмму распределения полученных значений контрольных сумм и хешей.