

# Основы построения файловых систем



## Статистика Backblaze по поломкам HDD в 2018 году

Поломки среди наиболее многочисленных дисков составили:

Модель	Число дисков	% поломавшихся
Seagate ST12000NM00007	25100	1.29%
Seagate ST8000NM0055	14400	0.70%
Seagate ST4000DM000	24300	2.28%

<https://www.backblaze.com/blog/2018-hard-drive-failure-rates/>

В 2015 наблюдались поломки 3.31% дисков от Seagate.

## Эксперимент в CERN о надёжности хранения данных

- Приложение пишет 1Gb данных на диск следующим образом:
  - Записать 1Mb,
  - Подождать 1с,
  - Повторить.
- Запускаем такое приложение на каждом из дисков на кластере из 3000 машин с HW RAID.
- Через 3 недели читаем содержимое файлов.

## Эксперимент в CERN о надёжности хранения данных

- Приложение пишет 1Gb данных на диск следующим образом:
  - Записать 1Mb,
  - Подождать 1с,
  - Повторить.
- Запускаем такое приложение на каждом из дисков на кластере из 3000 машин с HW RAID.
- Через 3 недели читаем содержимое файлов.
- Нашлось примерно 150 одномогабайтных блоков с изменившимся содержимым, причём чтение из них завершалось «успешно» с точки зрения как оборудования, так и файловой системы.

## Эксперимент в CERN о надёжности хранения данных

### Выводы:

- Данные нельзя хранить в единственном экземпляре,
- Необходимы контрольные суммы для проверки целостности,
- Необходима активная фоновая проверка данных.
- Хранение реплик или использование Reed-Solomon,
- ZFS и btrfs хранят криптографические хеши всех записанных данных, ext4 хранит только CRC,
- Online scrubbing & repair в ZFS и btrfs или в HW RAID-контроллерах.

## Обработка повреждений ФС в различных приложениях

В работе [1] данные следующих приложений располагали на ФС, случайно подменяющей содержимое блоков:

- Redis,
- ZooKeeper,
- Cassandra,
- Kafka,
- RethinkDB,
- LogCabin.

[1] <https://www.usenix.org/system/files/conference/fast17/fast17-ganesan.pdf>

[2] <https://www.usenix.org/system/files/conference/fast18/fast18-alagappan.pdf>

## Обработка повреждений ФС в различных приложениях

В работе [1] данные следующих приложений располагали на ФС, случайно подменяющей содержимое блоков:

- Redis,
  - Не проверяет контрольные суммы пользовательских данных,
  - Как следствие, реплицирует некорректные данные между узлами,
  - Повреждения в ФС обрабатываются с помощью assert().
- ZooKeeper,
- Cassandra,
- Kafka,
- RethinkDB,
- LogCabin.

[1] <https://www.usenix.org/system/files/conference/fast17/fast17-ganesan.pdf>

[2] <https://www.usenix.org/system/files/conference/fast18/fast18-alagappan.pdf>

## Обработка повреждений ФС в различных приложениях

В работе [1] данные следующих приложений располагали на ФС, случайно подменяющей содержимое блоков:

- Redis,
- ZooKeeper,
  - Проверяет наличие ошибки во всех данных, но делает это с помощью `assert()`,
  - Забыли рассмотреть случай, когда IO завершается неудачно и в заголовке транзакции, и в журнале,
  - Использует Adler32 для проверки целостности данных,
- Cassandra,
- Kafka,
- RethinkDB,
- LogCabin.

[1] <https://www.usenix.org/system/files/conference/fast17/fast17-ganesan.pdf>

[2] <https://www.usenix.org/system/files/conference/fast18/fast18-alagappan.pdf>



## Обработка повреждений ФС в различных приложениях

В работе [1] данные следующих приложений располагали на ФС, случайно подменяющей содержимое блоков:

- Redis,
- ZooKeeper,
- Cassandra,
  - Забыли о контрольных суммах для несжатых данных,
  - При несоответствии данных и контрольной суммы выбирает последнюю запись как правильную, поэтому может распространять повреждения между репликами,
- Kafka,
- RethinkDB,
- LogCabin.

[1] <https://www.usenix.org/system/files/conference/fast17/fast17-ganesan.pdf>

[2] <https://www.usenix.org/system/files/conference/fast18/fast18-alagappan.pdf>

## Способы проверки целостности данных

У нас упоминались два инструмента для проверки целостности данных:

- Cyclic redundancy checks,
- Криптографические хеш-суммы.

Обсудим их детальнее.

## Cyclic Redundancy Check

Рассмотрим сообщение как последовательность битов (элементов  $GF(2)$ ) 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  $d$ , посчитаем  $r(X)$  – остаток от деления  $M(X) * X^d$  на  $C(X)$ .

$r(X)$  называется CRC сообщения  $M$ .

Теперь построим многочлен

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

Ему соответствует сообщение  $a_{n-1}a_{n-2} \dots a_0$ , к которому дописали биты, равные коэффициентам  $r$  (внимание:  $r$  может быть степени меньше  $m$ , тогда считаем коэффициенты при старших степенях нулями).

CRC очень хорошо приспособлены для аппаратной реализации: из арифметических операций нужен 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 любую ошибку, изменяющую только один бит.
- Если в разложении  $C(X)$  на неприводимые множители есть многочлен степени  $m$ , то  $C(X)$  определяет любую ошибку, изменяющую только два бита, расположенных на расстоянии, меньшем  $m$ .
- Если  $C(X)$  делится на  $X+1$ , то он определяет любую ошибку, меняющую нечётное число бит.

## Cyclic Redundancy Check

Типичная схема применения CRC:

```
struct something
{
    some fields
    ...
    u64 crc;
}
```

1. Вычислить CRC всех полей структуры, кроме `something->crc`,
2. Записать в `something->crc` такое значение, чтобы CRC от всей структуры равнялся нулю.

**Упражнение:** пусть дано сообщение  $M$  и порождающий многочлен  $C(X)$  степени  $d$ . Найти целое  $d$ -битовое число  $X$  такое, что  $CRC(\text{concat}(M, X)) = 0$ .

# 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.

- Redis,
- ZooKeeper,
  - Проверяет наличие ошибки во всех данных, но делает это с помощью `assert()`,
  - Забыли рассмотреть случай, когда IO завершается неудачно и в заголовке транзакции, и в журнале,
  - Использует **Adler32** для проверки целостности данных,
- Cassandra,
- Kafka,
- RethinkDB,
- LogCabin.

Adler32 построен для определения ошибок архиваторов и годится только для коротких строк. Проверка больших блоков в ZooKeeper из-за этого ненадёжна и ZooKeeper может возвращать повреждённые данные.

## 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.

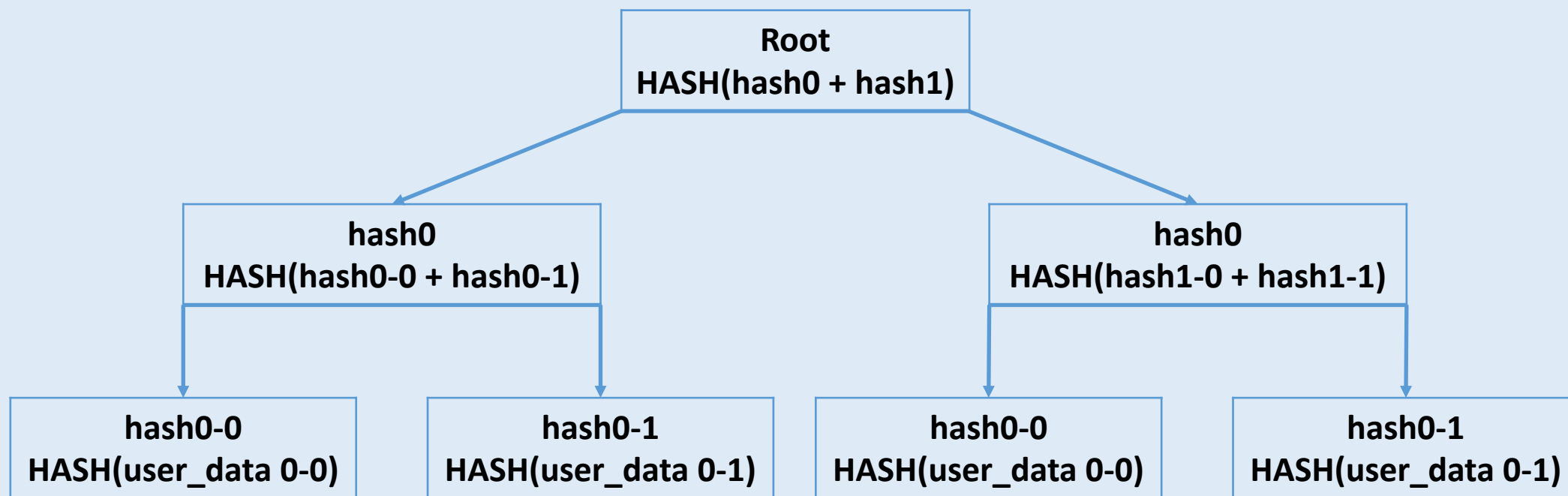


Примерная скорость вычисления хешей и CRC\*

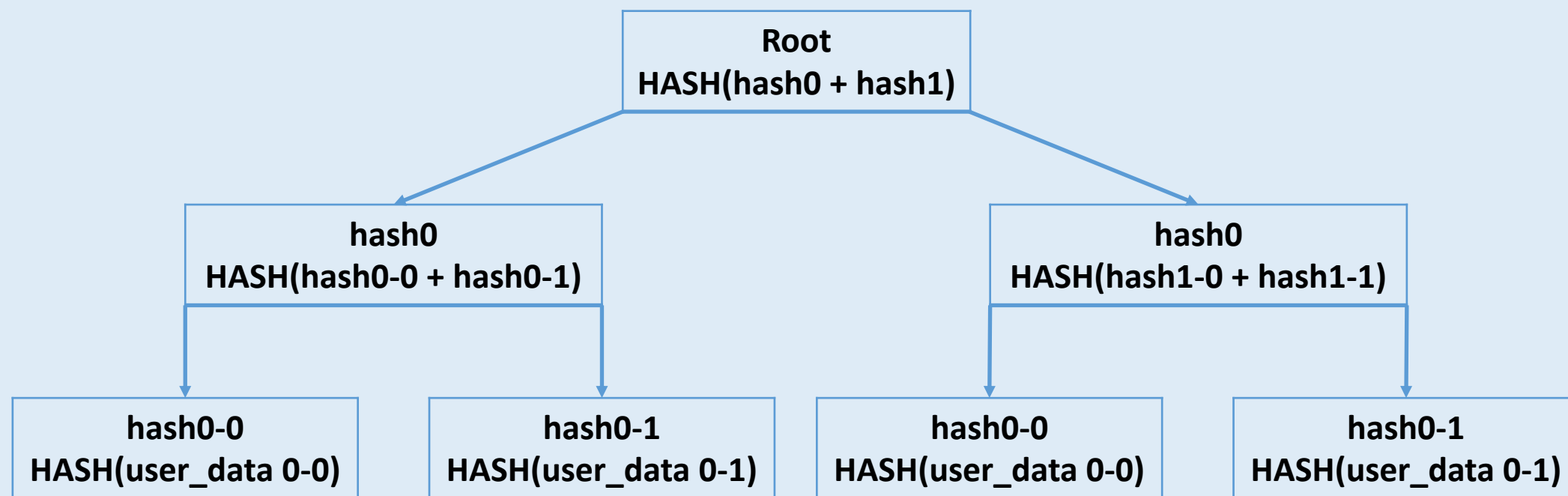
Алгоритм	GB/sec	Cycles/byte
CRC32	11.0	0.22
MD5	9.7	0.25
SHA-1	5.4	0.45
SHA-512	2.2	1.08

\* Данные для реализаций из ISA-L на Xeon 8180 (Neon City)

## Пример проверки целостности дерева: Merkle trees



# Пример проверки целостности дерева: Merkle trees



Применения:

- проверка целостности структуры дерева каталогов и дерева экстендов (ZFS, btrfs),
- проверка подлинности данных в p2p-сетях,
- быстрое определение частей деревьев, подлежащих синхронизации в распределённой БД (например, DynamoDB).

## RAID – Redundant Array of Independent (Inexpensive) Disks

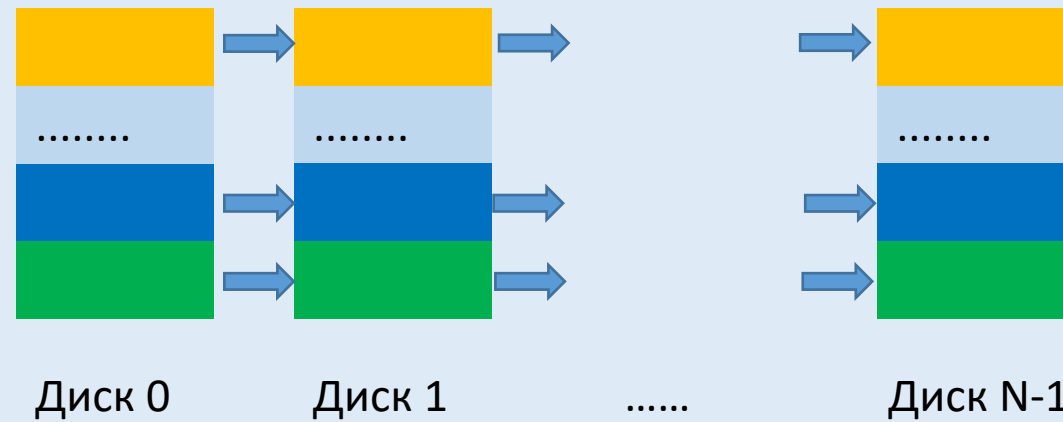
Для чего нужен:

- Большая надёжность, чем у отдельных дисков,
- Большая вместимость, чем у отдельных дисков.

## Уровни RAID

### RAID0 (stripe)

Данные разрезаются на последовательные куски длины  $N * B$ , каждый кусок разделяется на  $N$  частей, которые записываются на различные диски:



# Уровни RAID

RAID1 (mirror)

Каждый диск в массиве содержит одни и те же данные:

.....

Диск 0

==

.....

Диск 1

==

.....

==

.....

Диск N-1

# Уровни RAID

RAID4

Массив состоит из N+1 дисков. На первых N дисках данные хранятся, как на RAID0. На последнем диске каждый блок вычисляется как XOR соответствующих блоков на N дисках.

.....

b10

b00

→

.....

b11

b01

→

.....

.....

b10 + b11 + ...

b00 + b01 + ...

Диск 0

Диск 1

.....

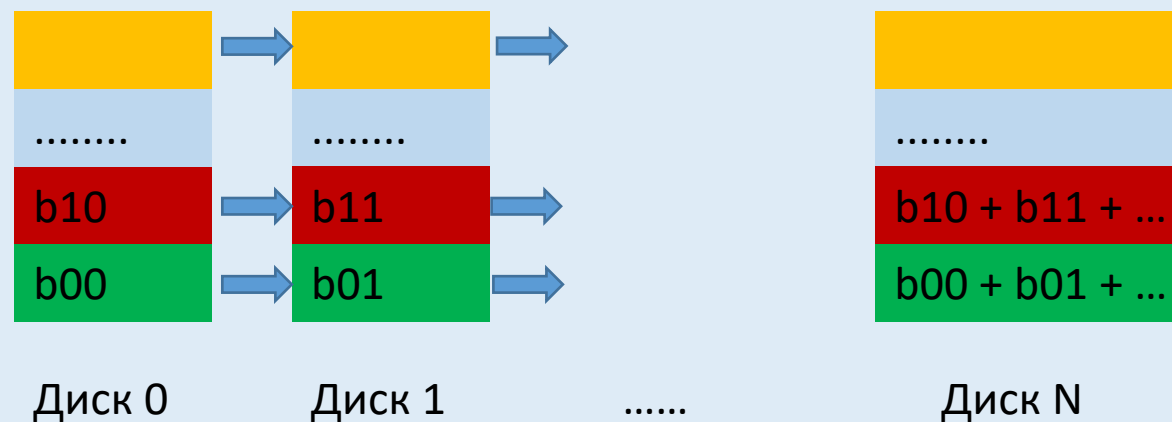
Диск N

При потере любого диска массив остаётся работоспособным.

# Уровни RAID

## RAID4

Массив состоит из  $N+1$  дисков. На первых  $N$  дисках данные хранятся, как на RAID0. На последнем диске каждый блок вычисляется как XOR соответствующих блоков на  $N$  дисках.



При потере любого диска массив остаётся работоспособным.

Такой массив имеет концептуальный недостаток: диск с блоками чётности будет изнашиваться быстрее других дисков.



# Уровни RAID

RAID5	<p>Массив строится так же, как и RAID4, но блоки чётности в разных страйпах хранятся на разных дисках:</p> <div><div><div></div><div>.....</div><div>b20</div><div>b10 + b11</div><div>b00</div></div><div><div></div><div>.....</div><div>b20 + b21</div><div>b10</div><div>b01</div></div><div><div></div><div>.....</div><div>b21</div><div>b11</div><div>b00 + b01</div></div></div>
-------	--

## Write holes

Запись на разные диски будет происходить в разное время.

Рассмотрим такой сценарий:

1. начинается запись на RAID1,
2. диск #0 обработал запрос на запись сектора,
3. произошёл сбой питания,
4. на диске #1 сектор остался без изменений.

## Write holes

Аппаратный способ решения:

- BBU (Battery Backup Unit) в RAID-контроллерах.

Программные способы решения:

- write intent bitmap (linux md),
- checksumming + COW (ZFS),
- SSD journal: <https://lwn.net/Articles/665299/> .

Write intent bitmap, помимо исправления write holes, позволяет уменьшить время проверки и перестроения массива после аварийного выключения.

## Скорость записи на RAID5

Из-за необходимости переживать аварийные выключения мы не имеем права одновременно изменять несколько блоков в одном страйпе. Значит, скорость записи на RAID5 получается такая же, как на одиночный диск.

## Ещё одна проблема RAID5

Восстановление данных занимает достаточно долго времени\*, притом в течение всего этого промежутка на оставшиеся диски создаётся высокая нагрузка, что повышает вероятность выхода из строя ещё одного диска во время перестроения RAID5.

В практике наблюдалось достаточно количество ситуаций, когда во время перестроения массива отказывал второй диск. По этой причине на больших массивах от RAID5 отказались в пользу RAID6, разрешающего терять произвольные два диска.

*\* Перезаписать диск 10Tb на скорости 100Mb/sec займёт порядка полутора суток.*