Reed – Solomon Coding for Fault-Tolerance in RAID-like Systems

Grigory Rozhkov, Dmitry Kharkovsky

28 February 2013

Постановка задачи

Пусть есть:

- n устройств хранения данных: $D_1, D_2, \ldots, D_n$, каждое из которых имеет вместимость k байт, *Data Devices*

Постановка задачи

Пусть есть:

- n устройств хранения данных: $D_1, D_2, \ldots, D_n$, каждое из которых имеет вместимость k байт, *Data Devices*
- Еще m устройств такой же вместимости, имеющих вспомогательный характер: $C_1, C_2, \ldots, C_m$, *Checksum Devices*

Постановка задачи

Пусть есть:

- n устройств хранения данных: $D_1, D_2, \ldots, D_n$, каждое из которых имеет вместимость k байт, *Data Devices*
- Еще m устройств такой же вместимости, имеющих вспомогательный характер: $C_1, C_2, \ldots, C_m$, *Checksum Devices*
- Мы, мечтающие о том, чтобы при утрате любых m устройств, было возможно восстановить всю информацию

RAID-like systems

- *RAID* = 'Redundant Arrays of Inexpensive Disks'

RAID-like systems

- *RAID* = 'Redundant Arrays of Inexpensive Disks'
- *RAID-like system* – примерно то же самое, но в терминах распределенных систем

RAID-like systems

- *RAID* = 'Redundant Arrays of Inexpensive Disks'
- *RAID-like system* – примерно то же самое, но в терминах распределенных систем
- Чем больше устройств хранения данных, тем меньше матожидание выхода из строя системы в совокупности

RAID-like systems

- *RAID* = 'Redundant Arrays of Inexpensive Disks'
- *RAID-like system* – примерно то же самое, но в терминах распределенных систем
- Чем больше устройств хранения данных, тем меньше матожидание выхода из строя системы в совокупности
- *RAID Level 5*, $m = 1$, одновременный выход из строя маловероятен

Общий смысл алгоритма

- Рассматривается случай полного выхода диска из строя, но не случай изменения данных

Общий смысл алгоритма

- Рассматривается случай полного выхода диска из строя, но не случай изменения данных
- Т.н. RS-Raid алгоритм разбивает каждый диск на *слова* (*words*) по w бит, где w выбирается программистом
- На каждом диске $l = (k\ bytes)\left(\frac{8\ bits}{byte}\right)\left(\frac{1\ word}{w\ bits}\right) = \frac{8k}{w}$ слов

Общий смысл алгоритма

- Рассматривается случай полного выхода диска из строя, но не случай изменения данных
- Т.н. RS-Raid алгоритм разбивает каждый диск на *слова* (*words*) по w бит, где w выбирается программистом
- На каждом диске $l = (k\ bytes)\left(\frac{8\ bits}{byte}\right)\left(\frac{1\ word}{w\ bits}\right) = \frac{8k}{w}$ слов
- Слова на C_i суть результаты некой функции F_i, примененной к словам из $D_1, \ldots, D_n$

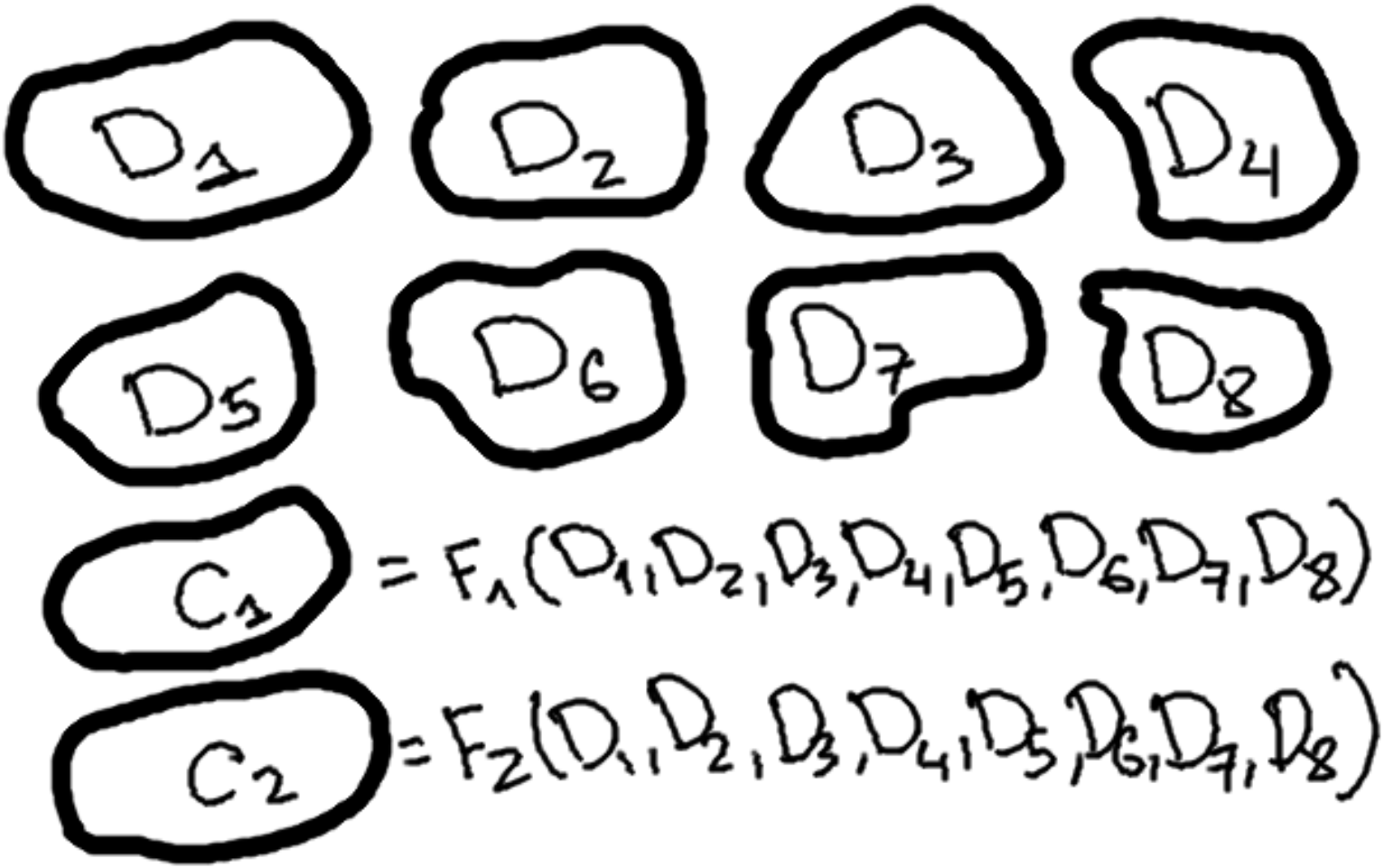
Общий смысл алгоритма

- Рассматривается случай полного выхода диска из строя, но не случай изменения данных
- Т.н. RS-Raid алгоритм разбивает каждый диск на *слова* (*words*) по w бит, где w выбирается программистом
- На каждом диске $l = (k\ bytes)\left(\frac{8\ bits}{byte}\right)\left(\frac{1\ word}{w\ bits}\right) = \frac{8k}{w}$ слов
- Слова на C_i суть результаты некой функции F_i, примененной к словам из $D_1, \ldots, D_n$
- Например, если $l = 1$ для всех дисков, то $c_i = F_i(d_1, d_2, \ldots, d_n)$

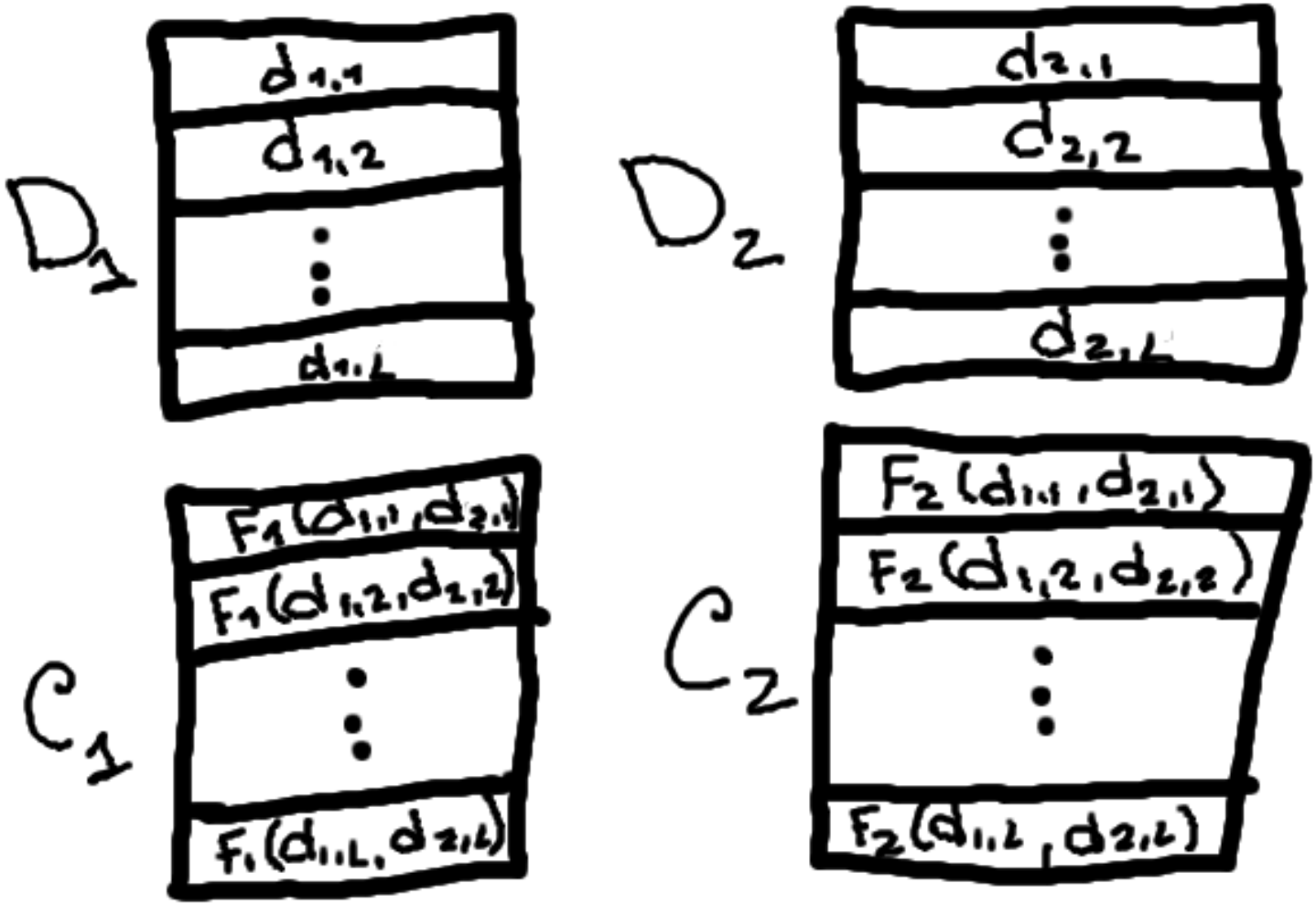
Общий смысл алгоритма

- Рассматривается случай полного выхода диска из строя, но не случай изменения данных
- Т.н. RS-Raid алгоритм разбивает каждый диск на *слова* (*words*) по w бит, где w выбирается программистом
- На каждом диске $l = (k\ bytes)\left(\frac{8\ bits}{byte}\right)\left(\frac{1\ word}{w\ bits}\right) = \frac{8k}{w}$ слов
- Слова на C_i суть результаты некой функции F_i, примененной к словам из $D_1, \ldots, D_n$
- Например, если $l = 1$ для всех дисков, то $c_i = F_i(d_1, d_2, \ldots, d_n)$
- Функция пересчета: $c_i^{new} = G_{i,j}(d_j, d_j^{new}, c_i)$

Художества


D1
D2
D3
D4
D5
D6
D7
D8
C1
= F1(D1,D2,D3,D4,D5,D6,D7,D8)
C2
= F2(D1,D2,D3,D4,D5,D6,D7,D8)

Художества


$d_{1,1}$
$d_{1,2}$
$d_{1,L}$
D_1
$d_{2,1}$
$d_{2,2}$
$d_{2,L}$
D_2
$F_1(d_{1,1}, d_{2,1})$
$F_1(d_{1,2}, d_{2,2})$
$F_1(d_{1,L}, d_{2,L})$
C_1
$F_2(d_{1,1}, d_{2,1})$
$F_2(d_{1,2}, d_{2,2})$
$F_2(d_{1,L}, d_{2,L})$
C_2

Общий смысл алгоритма, пример

Пусть $m = 1$, $w = 1$.

В качестве функции F_1 рассмотрим XOR:

$$c_1 = F_1(d_1, \ldots, d_n) = d_1 \oplus d_2 \oplus \cdots \oplus d_n$$

Общий смысл алгоритма, пример

Пусть $m = 1$, $w = 1$.
В качестве функции F_1 рассмотрим XOR:

$$c_1 = F_1(d_1, \ldots, d_n) = d_1 \oplus d_2 \oplus \cdots \oplus d_n$$

Если какое-то слово d_j изменило свое значение на d_j^{new}, то c_1 пересчитывается следующим образом:

$$c_1^{new} = G_{1,j} = c_1 \oplus d_j \oplus d_j^{new}$$

Общий смысл алгоритма, пример

Пусть $m = 1$, $w = 1$.
В качестве функции F_1 рассмотрим XOR:

$$c_1 = F_1(d_1, \ldots, d_n) = d_1 \oplus d_2 \oplus \cdots \oplus d_n$$

Если какое-то слово d_j изменило свое значение на d_j^{new}, то c_1 пересчитывается следующим образом:

$$c_1^{new} = G_{1,j} = c_1 \oplus d_j \oplus d_j^{new}$$

В случае если один из дисков, например D_j, прекратил функционировать, значение восстанавливается следующим образом:

$$d_j = d_1 \oplus \cdots \oplus d_{j-1} \oplus d_{j+1} \oplus \ldots d_n \oplus c_1$$

Математический аппарат

- Матрица Вандермонда
- Метод Гаусса
- Поля Галуа aka конечные поля

RS-Raid алгоритм

Рассматриваем линейные функции F_i:

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j f_{i,j}$$

RS-Raid алгоритм

Рассматриваем линейные функции F_i:

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j f_{i,j}$$

В векторном виде:

$$FD = C$$

RS-Raid алгоритм

Рассматриваем линейные функции F_i:

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j f_{i,j}$$

В векторном виде:

$$FD = C$$

F – $m \times n$ матрица Вандермонда: $f_{i,j} = j^{i-1}$

RS-Raid алгоритм

Рассматриваем линейные функции F_i:

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j f_{i,j}$$

В векторном виде:

$$FD = C$$

F – $m \times n$ матрица Вандермонда: $f_{i,j} = j^{i-1}$
Функция пересчета очевидно выглядит следующим образом:

$$c_i^{new} = G_{i,j}(d_j, d_j^{new}, c_i) = c_i + f_{i,j}(d_j^{new} - d_j)$$

Восстановление после сбоев

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \dots & n^{m-1} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$$

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах
- Вводятся поля Галуа – поля мощности 2^w

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах
- Вводятся поля Галуа – поля мощности 2^w
- Важное требование: $2^w > n + m$

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах
- Вводятся поля Галуа – поля мощности 2^w
- Важное требование: $2^w > n + m$
- Элементы обозначаются числами от 0 до $2^w - 1$

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах
- Вводятся поля Галуа – поля мощности 2^w
- Важное требование: $2^w > n + m$
- Элементы обозначаются числами от 0 до $2^w - 1$
- Сложение – обычный XOR, например в $GF(2^4)$
  $11 + 7 = 1101 \oplus 0111 = 1100 = 12$

Арифметика в конечных полях

- Алгоритм Гаусса требует всякие операции, которые непонятно что значат на словах
- Вводятся поля Галуа – поля мощности 2^w
- Важное требование: $2^w > n + m$
- Элементы обозначаются числами от 0 до $2^w - 1$
- Сложение – обычный XOR, например в $GF(2^4)$
  $11 + 7 = 1101 \oplus 0111 = 1100 = 12$
- Вычитание совпадает со сложением

Арифметика в конечных полях

- Умножение и деление

Арифметика в конечных полях

- Умножение и деление
- *intgflog*[] – таблица логарифмов для чисел от 1 до $2^w - 1$
- *intgfilog*[] – таблица инвертированного логарифма для чисел от 0 до $2^w - 2$

Арифметика в конечных полях

- Умножение и деление
- $intgflog[]$ – таблица логарифмов для чисел от 1 до $2^w - 1$
- $intgfilog[]$ – таблица инвертированного логарифма для чисел от 0 до $2^w - 2$
- $gfilog[gflog[i]] == i, gflog[gfilog[i]] == i$

Арифметика в конечных полях

- Умножение и деление
- $intgflog[]$ – таблица логарифмов для чисел от 1 до $2^w - 1$
- $intgfilog[]$ – таблица инвертированного логарифма для чисел от 0 до $2^w - 2$
- $gfilog[gflog[i]] == i, gflog[gfilog[i]] == i$
- $3 * 7 = gfilog[gflog[3] + gflog[7]] = gfilog[4 + 10] = gfilog[14] = 9$
- $3 \div 7 = gfilog[gflog[3] - gflog[7]] = gfilog[4 - 10] = gfilog[9] = 14$

Таблица логарифма и инвертированного логарифма

i	0	1	2	3	4	5	6	7
$gflog[i]$	–	0	1	4	2	8	5	10
$gfilog[i]$	1	2	4	8	3	6	12	11
i	8	9	10	11	12	13	14	15
$gflog[i]$	3	14	9	7	6	13	11	12
$gfilog[i]$	5	10	7	14	15	13	9	–

Таблицы вычисляются с помощью полиномов

w	polynome
4	$x^4 + x + 1$
8	$x^8 + x^4 + x^3 + x^2 + 1$
16	$x^{16} + x^{12} + x^3 + x + 1$
32	$x^{32} + x^{22} + x^2 + x + 1$
64	$x^{64} + x^4 + x^3 + x + 1$

Итого

1. Выбираем число w такое, что $2^w > n + m$. Лучше всего 8 или 16. При $w = 16$ сможем поддерживать до 65535 дисков

Итого

1. Выбираем число w такое, что $2^w > n + m$. Лучше всего 8 или 16. При $w = 16$ сможем поддерживать до 65535 дисков
2. Заводим таблицы *gflog* и *gfilog*

Итого

1. Выбираем число w такое, что $2^w > n + m$. Лучше всего 8 или 16. При $w = 16$ сможем поддерживать до 65535 дисков
2. Заводим таблицы *gflog* и *gfilog*
3. Заводим $m \times n$ матрицу Вандермонда, где возведение в степень выполнено в $GF(2^w)$

Итого

1. Выбираем число w такое, что $2^w > n + m$. Лучше всего 8 или 16. При $w = 16$ сможем поддерживать до 65535 дисков
2. Заводим таблицы *gflog* и *gfilog*
3. Заводим $m \times n$ матрицу Вандермонда, где возведение в степень выполнено в $GF(2^w)$
4. Вычисляем значения слов на C_i

Итого

1. Выбираем число w такое, что $2^w > n + m$. Лучше всего 8 или 16. При $w = 16$ сможем поддерживать до 65535 дисков
2. Заводим таблицы *gflog* и *gfilog*
3. Заводим $m \times n$ матрицу Вандермонда, где возведение в степень выполнено в $GF(2^w)$
4. Вычисляем значения слов на C_i
5. Если умерло $< m$ дисков, восстанавливаем информацию с помощью матрично-векторного уравнения, решенного методом Гаусса

Пример заполнения таблицы

Listing 1: Table filling

```
int prim_poly = 19;
int x_to_w = 1 << w;
int *gflog, gfilog = new int[x_to_w];

int b = 1;
for (log = 0; log < x_to_w - 1; log++ ){
gflog[b] = log;
gfilog[log] = b;
b = b << 1;
if (b & x_to_w) b = b ^ prim_poly
}
```

Пример

Пусть $m = 3$ и $n = 3$. Возьмем $w = 4$.
Пусть $D_1 = 3$, $D_2 = 13$ и $D_3 = 9$ Матрица F выглядит так:

$$\begin{pmatrix} 1^0 & 2^0 & 3^0 \\ 1^1 & 2^1 & 3^1 \\ 1^2 & 2^2 & 3^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

Пример

$$c_1 = (1)(3) \oplus (1)(13) \oplus (1)(9) = 3 \oplus 13 \oplus 9 =$$

$$= 0011 \oplus 1101 \oplus 1001 = 0111 = 7$$

$$c_2 = (1)(3) \oplus (2)(13) \oplus (3)(9) = 3 \oplus 9 \oplus 8 =$$

$$= 0011 \oplus 1001 \oplus 1000 = 0010 = 2$$

$$c_3 = (1)(3) \oplus (4)(13) \oplus (5)(9) = 3 \oplus 1 \oplus 11 =$$

$$= 0011 \oplus 0001 \oplus 1011 = 10001 = 9$$

Пример

Поменяли $D_2 = 1$. Тогда изменение
$D_2 = 1 - 13 = 0001 \oplus 1101 = 12$

$$c_1 = 7 \ominus (1)(12) = 0111 \oplus 1100 = 11$$

$$c_2 = 2 \ominus (2)(12) = 2 \oplus 11 = 0010 \oplus 1011 = 9$$

$$c_3 = 9 \ominus (4)(12) = 9 \oplus 5 = 10001 \oplus 0101 = 12$$

Пример

Пусть сломались D_2, D_3 и C_1. Удалим из матрицы:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot D = \begin{pmatrix} 3 \\ 11 \\ 9 \end{pmatrix}$$

$$\mathrm{D}= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 3 & 1 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 11 \\ 9 \end{pmatrix}$$

Пример

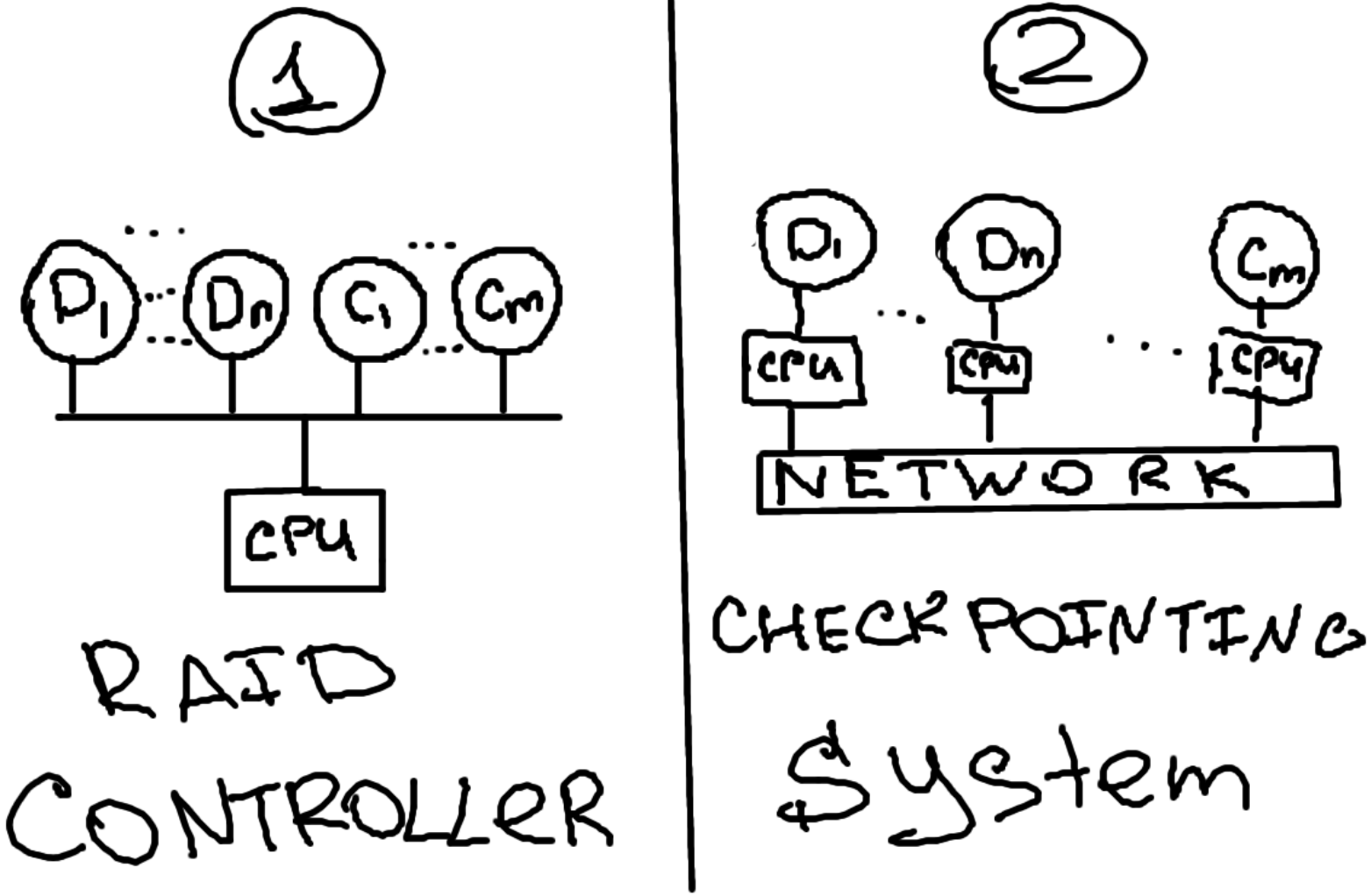
$$D_2 = (2)(3) \oplus (3)(11) \oplus (1)(9) = 6 \oplus 14 \oplus 9 = 1$$

$$D_3 = (3)(3) \oplus (2)(11) \oplus (1)(9) = 5 \oplus 5 \oplus 9 = 9$$

$$C_3 = (1)(3) \oplus (4)(1) \oplus (5)(9) = 3 \oplus 4 \oplus 11 = 12$$

Реализация и производительность

Две реализации:



RAID controllers. Запись

- Разбиваем файл на n частей, необходимо вычислить все c_i

RAID controllers. Запись

- Разбиваем файл на n частей, необходимо вычислить все c_i
- Вычисление c_1 потребует $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}\right)$

RAID controllers. Запись

- Разбиваем файл на n частей, необходимо вычислить все c_i
- Вычисление c_1 потребует $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}\right)$
- А вычисление всех остальных – $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}+\frac{1}{R_{GFmult}}\right)$

RAID controllers. Запись

- Разбиваем файл на n частей, необходимо вычислить все c_i
- Вычисление c_1 потребует $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}\right)$
- А вычисление всех остальных – $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}+\frac{1}{R_{GFmult}}\right)$
- Таким образом вычисление всех m контрольных блоков потребует $S_{Block}(n-1)\left(\frac{m}{R_{XOR}}+\frac{m-1}{R_{GFmult}}\right)$

RAID controllers. Запись

- Разбиваем файл на n частей, необходимо вычислить все c_i
- Вычисление c_1 потребует $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}\right)$
- А вычисление всех остальных – $S_{Block}(n-1)\left(\frac{1}{R_{XOR}}+\frac{1}{R_{GFmult}}\right)$
- Таким образом вычисление всех m контрольных блоков потребует $S_{Block}(n-1)\left(\frac{m}{R_{XOR}}+\frac{m-1}{R_{GFmult}}\right)$
- Еще необходимо учесть затраты на саму запись

RAID controllers. Обновление

- Необходимо вычислить $c_i + f_{i,j}(d_j^{new} - d_j)$

RAID controllers. Обновление

- Необходимо вычислить $c_i + f_{i,j}(d_j^{new} - d_j)$
- Вычитание – 1 XOR, умножение происходит 0 или 1 раз, сложение – еще один XOR

RAID controllers. Обновление

- Необходимо вычислить $c_i + f_{i,j}(d_j^{new} - d_j)$
- Вычитание – 1 XOR, умножение происходит 0 или 1 раз, сложение – еще один XOR
- Итого: $(m+1) \cdot writes + \frac{m+1}{R_{XOR}} + (1 - \delta_{j,1}) \frac{m-1}{R_{GFmult}}$

RAID controllers. Обновление

- Необходимо вычислить $c_i + f_{i,j}(d_j^{new} - d_j)$
- Вычитание – 1 XOR, умножение происходит 0 или 1 раз, сложение – еще один XOR
- Итого: $(m+1) \cdot writes + \frac{m+1}{R_{XOR}} + (1 - \delta_{j,1}) \frac{m-1}{R_{GFmult}}$
- *update penalty* by Gibson

RAID controllers. Восстановление

- Вышло из строя $y \leq m$ дисков

RAID controllers. Восстановление

- Вышло из строя $y \leq m$ дисков
- Алгоритм Гаусса сработает за $\mathbb{O}(y^2 n)$

RAID controllers. Восстановление

- Вышло из строя $y \leq m$ дисков
- Алгоритм Гаусса сработает за $\mathbb{O}(y^2 n)$
- Стоимость восстановления одного блока:

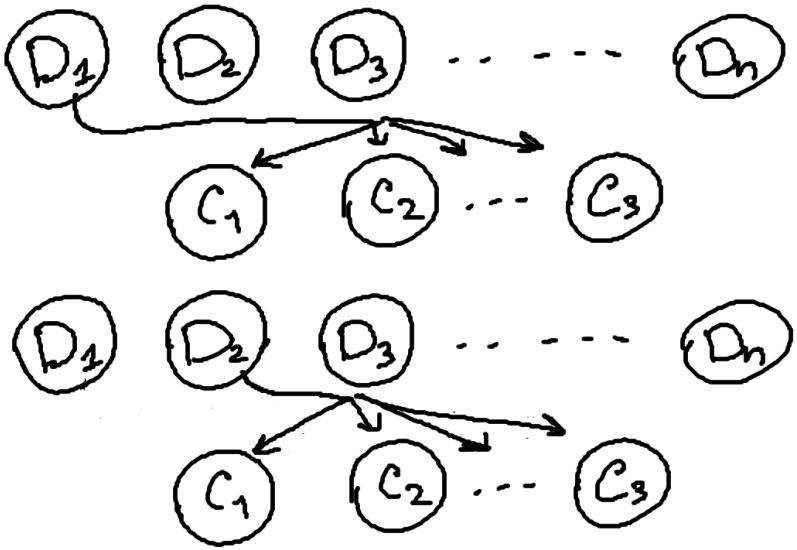
$$n \cdot reads + \frac{yS_{Block}(n-1)}{R_{XOR}} + \frac{yS_{Block}n}{R_{GFmult}} + y \cdot writes$$

Checkpointing systems

- Broadcast Algorithm

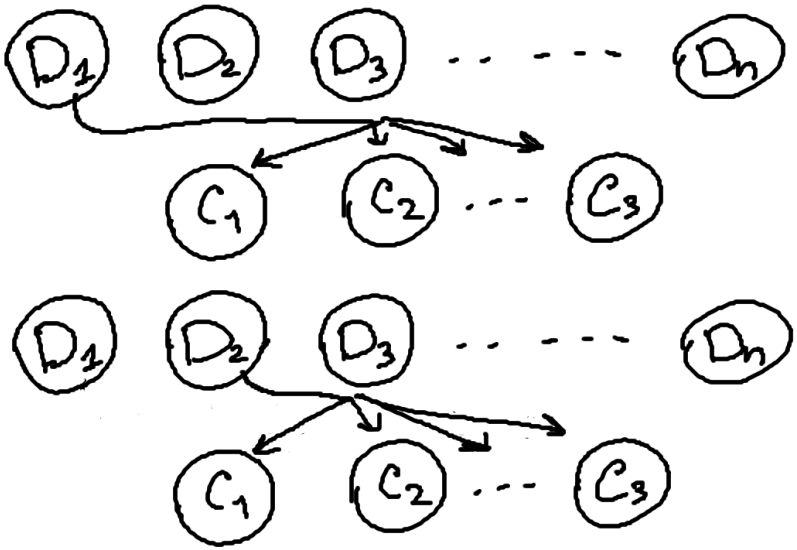
Checkpointing systems

- Broadcast Algorithm



Checkpointing systems

- Broadcast Algorithm



-
- На инициализацию потребуется

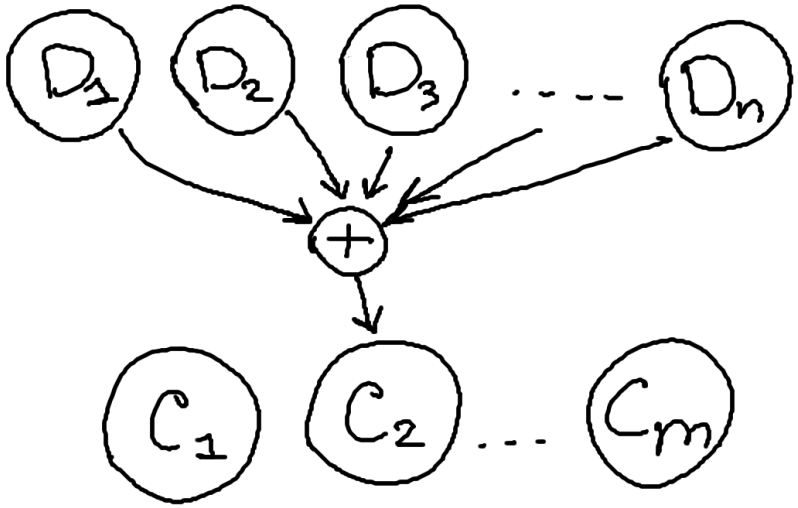
$$nS_{device}\left(\frac{1}{R_{broadcasting}}+\frac{1}{R_{GFmult}}+\frac{1}{R_{XOR}}\right)$$

Checkpointing systems

- The Fan-in Algorithm

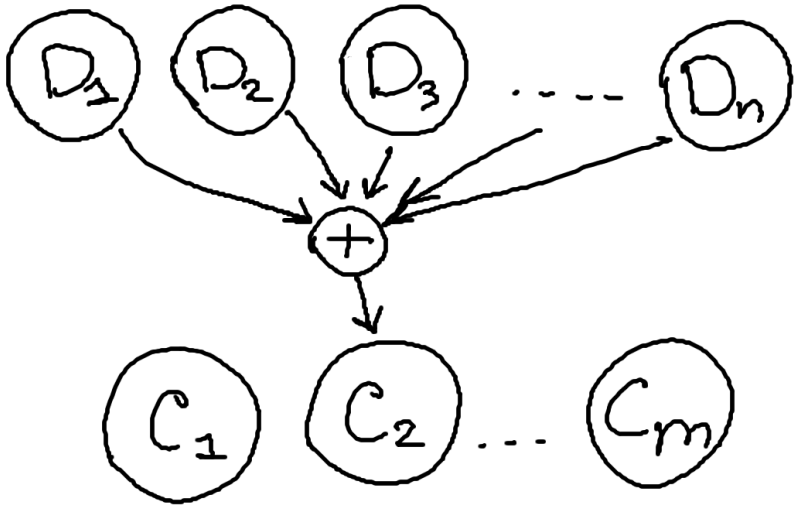
Checkpointing systems

- The Fan-in Algorithm



Checkpointing systems

- The Fan-in Algorithm



-
- На инициализацию потребуется

$$mS_{device}\left(\frac{log\ n}{R_{XOR}}+\frac{log\ n+1}{R_{network}}\right)+\left(\frac{(m-1)S_{device}}{R_{GFmult}}\right)$$

Checkpointing systems

Вторая важная операция – восстановление действует примерно так же и, т.к. метод Гаусса работает достаточно быстро, выполнять его можно на CPU каждого устройства, а не как-то распределенно.

Checkpointing systems

Вторая важная операция – восстановление действует примерно так же и, т.к. метод Гаусса работает достаточно быстро, выполнять его можно на CPU каждого устройства, а не как-то распределенно.
Выбор алгоритма зависит от проекта и целиком за Вами:)

Заключение

- Работает!

Заключение

- Работает!
- Нигде не сказано, что это лучший способ

Заключение

- Работает!
- Нигде не сказано, что это лучший способ
- И действительно, это не так

Заключение

- Работает!
- Нигде не сказано, что это лучший способ
- И действительно, это не так
- Легко реализовать
- Поддерживает до 65535 дисков в условиях предыдущего пункта

Спасибо

Спасибо за внимание!