



51,00

Рейтинг

YADRO

Discover. Design. Develop.

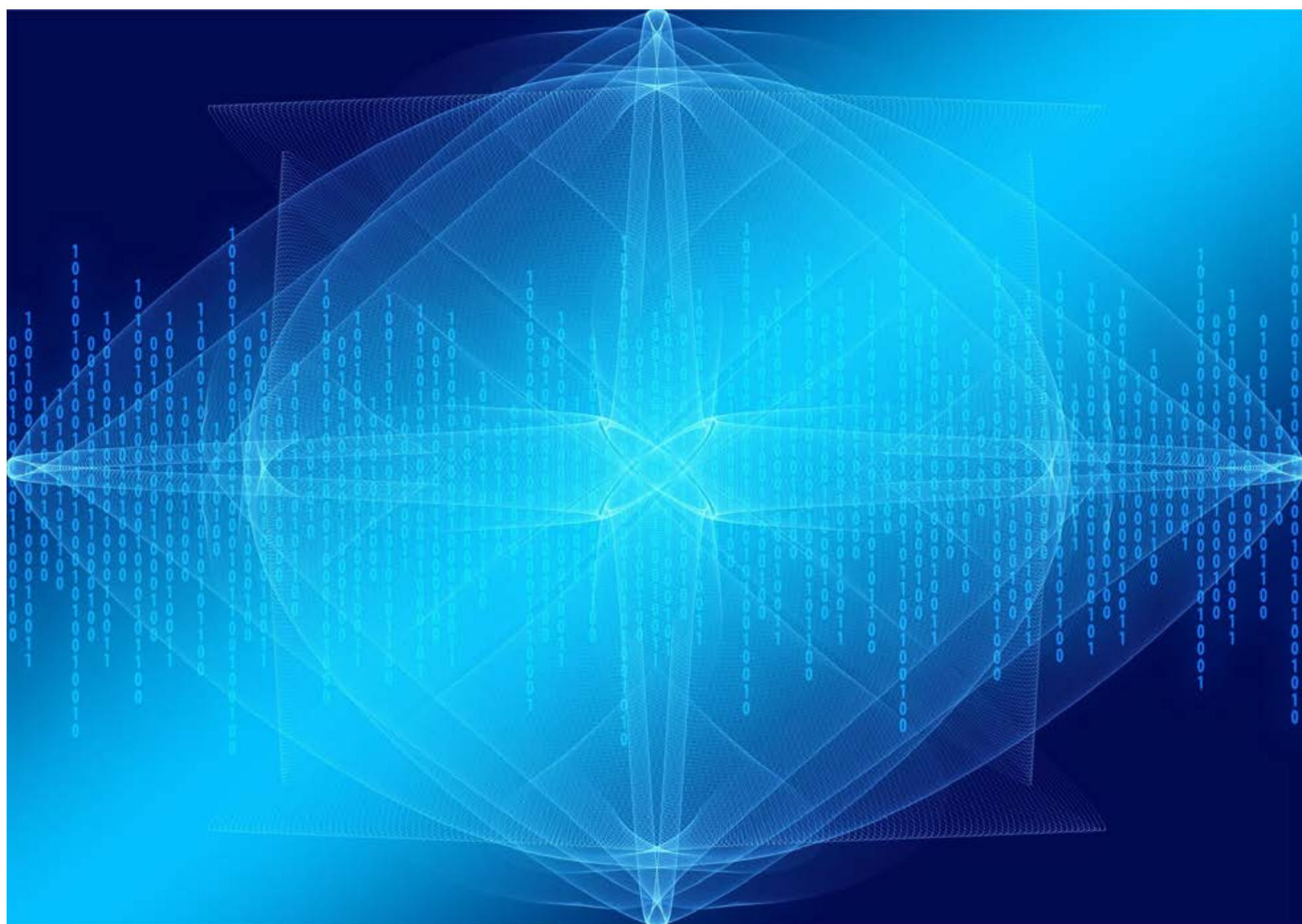


m-a-k-s-i-m 24 августа 2017 в 07:33

Коды Рида-Соломона. Часть 1 — теория простым языком

Блог компании YADRO, Алгоритмы

Добрый день! Меня зовут Максим, в YADRO, кроме всего прочего, я занимаюсь разработкой подсистемы, отвечающей за надежное хранение данных. Готовлю небольшой цикл статей про коды Рида-Соломона — теоретическую основу, практическую реализацию, применяемые на практике программные и аппаратные оптимизации. На Хабре и в остальной сети есть хорошие статьи по вопросам этой области — но по ним сложно разобраться, если ты новичок в теме. В этой статье я попытаюсь дать понятное введение в коды Рида-Соломона, а в следующих выпусках напишу, как все это запрограммировать.



Почему мы занимаемся кодами Рида-Соломона

Одно из главных направлений работы компании YADRO — разработка систем хранения данных (СХД). Можно долго обсуждать, какие характеристики данных систем являются самыми важными с точки зрения конечного пользователя. Это может быть скорость операций ввода/вывода, стоимость владения системой, уровень энергопотребления или что-то ещё. Но все эти характеристики будут иметь смысл только в том случае, если система обеспечивает необходимый пользователю уровень сохранности данных. Как результат, для всех разработчиков подобных систем, вопросы обеспечения надёжности хранения данных выходят на первый план.

Долгое время главным игроком в сфере безопасного хранения данных оставалась технология RAID. Подобные классические решения, наряду с известными плюсами, обладают и определёнными ограничениями. К ним можно отнести:

- относительно низкий уровень защиты (например, RAID 6 поддерживает выход из строя максимум двух дисков)

- необходимость держать запасные диски
- невозможность контролировать распределение «parity» блоков (это может быть важно для определенных типов дисков)
- сложность поддержки «гетерогенных» дисков

В связи с этим, в последнее время, набирают популярность технологии, основанные на всевозможных избыточных кодах (в англоязычной литературе за подобными технологиями закрепилось название — erasure coding), которые предоставляют гораздо больше возможностей по сравнению с классическим RAID.

На данный момент наша команда занимается разработкой системы хранения данных [TATLIN](#). По архитектуре системы мы планируем написать отдельную серию статей, тут лишь скажем, что в какой-то момент времени перед нами встала задача реализации собственной системы надёжного хранения данных. После изучения вопроса мы решили остановиться на проверенных временем кодах Рида-Соломона. Существуют готовые библиотеки (например, [ISA-L](#) от Intel или [Jerasure](#) авторства James S. Plank). В них реализованы соответствующие процедуры кодирования/декодирования. Но учитывая, что наша аппаратная платформа построена на базе архитектуры OpenPOWER, а вся логика системы реализована как модуль ядра ОС Linux, — было принято решение делать свою реализацию, оптимизированную под особенности нашей системы.

В этой первой статье я попытаюсь объяснить, о чем идет речь, когда говорят о кодах Рида-Соломона в контексте систем хранения данных. Начнем с простых примеров, которые позволят лучше уяснить сущность решаемых проблем.

Базовая задача восстановления утраченного

Предположим, что есть два произвольных целых числа A и B . Необходимо разработать алгоритм, который двум этим числам поставит в соответствие третье число R , причём такое, что по нему будет возможно однозначно восстановить любое из двух исходных чисел при потере одного из них. Другими словами, имея пару $(A$ и $R)$ или $(B$ и $R)$, возможно однозначно восстановить B или A соответственно. Существует много вариантов реализации подобного алгоритма, но, вероятно, самым простым является тот, который к двум исходным числам применяет битовую операцию «Исключающее «ИЛИ» (XOR)», т.е. $R = A \text{ xor } B$. Тогда для восстановления A можно воспользоваться равенством $A = R \text{ xor } B$ (для восстановления B , $B = R \text{ xor } A$).

Немного усложним задачу. Предположим, что исходных чисел не два, а три — A , B и C . Как и прежде, нужен алгоритм, позволяющий справиться с потерей одного из исходных чисел. Легко заметить, что ничто не мешает снова использовать битовую операцию «Исключающее «ИЛИ»», $R = A \text{ xor } B \text{ xor } C$. Для восстановления A можно воспользоваться равенством $A = R \text{ xor } B \text{ xor } C$.

Усложним задачу ещё немного. Как и прежде, имеется три числа A , B и C , а вот потерять, в этот раз, мы можем не одно, а любые два из них. Очевидно, что просто использовать операцию «Исключающее «ИЛИ» уже не получится. Как же восстанавливать потерянные числа?

Принцип кодирования Рида-Соломона на простых примерах с картинками

Одним, из возможных путей решения данной задачи, является применение кодов Рида-Соломона. Формальное описание метода легко найти в Интернете, но основная проблема заключается в том, что людям, далеким от «полей Галуа», достаточно сложно понять, как, собственно, всё это работает. Вот, например, [определение из Википедии](#):

Формальное описание [\[править | править вики-текст \]](#)

Коды Рида — Соломона являются важным частным случаем БЧХ-кода, корни порождающего полинома которого лежат в том же поле, над которым строится код ($m = 1$). Пусть α — элемент поля $GF(q)$ порядка n . Если α — примитивный элемент, то его порядок равен $q - 1$, то есть $\alpha^{q-1} = 1$, $\alpha^i \neq 1, 0 < i < q - 1$. Тогда нормированный полином $g(x)$ минимальной степени над полем $GF(q)$, корнями которого являются $d - 1$ подряд идущих степеней $\alpha^{l_0}, \alpha^{l_0+1}, \dots, \alpha^{l_0+d-2}$ элемента α , является порождающим полиномом кода Рида — Соломона над полем $GF(q)$:

$$g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \dots (x - \alpha^{l_0+d-2})$$

где l_0 — некоторое целое число (в том числе 0 и 1), с помощью которого иногда удается упростить кодер. Обычно полагается $l_0 = 1$. Степень многочлена $g(x)$ равна $d - 1$.

Попробуем разобраться с тем, как это работает, начав с более интуитивных вещей. Для этого вернемся к нашей последней задаче. Напомним, что есть три произвольных целых числа, любые два из них могут быть потеряны, необходимо научиться восстанавливать потерянные числа по оставшимся. Для этого применим «алгебраический» подход.

Но прежде необходимо напомнить еще об одном важном моменте. Технологии восстановления данных неспроста называются

методами избыточного кодирования. По исходным данным вычисляются некоторые «избыточные», которые потом позволяют восстановить потерянные. Не вдаваясь в подробности заметим, что эмпирическое правило такое — чем больше данных может быть потеряно, тем больше «избыточных» необходимо иметь. В нашем случае для восстановления двух чисел, нам придётся по некоторому алгоритму сконструировать два «избыточных». В общем случае, если нужно поддерживать потерю K чисел, необходимо соответственно иметь K избыточных.

Упомянутый выше «алгебраический» подход состоит в следующем. Составляется матрица специального вида размера 5×3 . Первые три строки этой матрицы образуют единичную матрицу, а последние две — это некоторые числа, о выборе которых мы поговорим позднее. В англоязычной литературе эту матрицу обычно называют *generator matrix*, в русскоязычной встречается несколько названий, в этой статье будет использоваться — порождающая матрица. Умножим сконструированную матрицу на вектор, составленный из исходных чисел A , B и C .

1	0	0
0	1	0
0	0	1
X_{00}	X_{01}	X_{02}
X_{10}	X_{11}	X_{12}

*

A
B
C

=

A
B
C
X_0
X_1

В результате умножения матрицы на вектор с данными получаем два «избыточных» числа, обозначенных на рисунке как X_0 и X_1 . Давайте посмотрим, как с помощью этих «избыточных» данных можно восстановить, к примеру, потерянные A и B .

Вычеркнем из порождающей матрицы строки, соответствующие «пропавшим» данным. В нашем случае A соответствует первой строке, а B – второй. Полученную матрицу умножим на вектор с данными, и в результате получим следующее уравнение:

0	0	1
X_{00}	X_{01}	X_{02}
X_{10}	X_{11}	X_{12}

*

B
C

=

X_0
X_1

Проблема лишь в том, что A и B мы потеряли, и они нам теперь неизвестны. Как мы можем их найти? Есть очень элегантное решение этой проблемы.

Вычеркнем соответствующие строки из порождающей матрицы и найдём обратную к ней. На рисунке эта обратная матрица обозначена как $\{Y_{ij}\}$. Теперь домножим левую и правую части исходного уравнения на эту обратную матрицу:

Y_{00}	Y_{01}	Y_{02}
Y_{10}	Y_{11}	Y_{12}
Y_{20}	Y_{21}	Y_{22}

*

0	0	1
X_{00}	X_{01}	X_{02}
X_{10}	X_{11}	X_{12}

*

A
B
C

=

Y_{00}	Y_{01}	Y_{02}
Y_{10}	Y_{11}	Y_{12}
Y_{20}	Y_{21}	Y_{22}

*

C
X_0
X_1

Сокращая матрицы в левой части уравнения (произведение обратной и прямой матриц есть единичная матрица), и учитывая тот факт, что в правой части уравнения нет неизвестных параметров, получаем выражения для искомых A и B .

A
B
C

=

Y_{00}	Y_{01}	Y_{02}
Y_{10}	Y_{11}	Y_{12}
Y_{20}	Y_{21}	Y_{22}

*

C
X_0
X_1

Собственно говоря, то, что мы сейчас проделали — основа всех типов кодов Рида-Соломона, применяемых в системах хранения данных. Процесс кодирования заключается в нахождении «избыточных» данных X_0 , X_1 , а процесс декодирования — в нахождении обратной матрицы и умножения её на вектор «сохранившихся» данных.

Нетрудно заметить, что рассмотренная схема может быть обобщена на произвольное количество «исходных» и «избыточных» данных. Другими словами, по исходным N числам можно построить K избыточных, причем всегда возможно восстановить потерю любых K из $N + K$ чисел. В этом случае порождающая матрица будет иметь размер $(N + K) \times N$, а верхняя часть матрицы размером $N \times N$ будет единичной.

Вернемся к вопросу о построении порождающей матрицы. Можем ли мы выбрать числа X_{ij} произвольным образом? Ответ – нет. Выбирать их нужно таким образом, чтобы вне зависимости от вычеркиваемых строк, матрица оставалась обратимой. К счастью, в математике давно известны типы матриц, обладающие нужным нам свойством. Например, [матрица Коши](#). В этом случае сам метод кодирования часто называют методом Коши-Рида-Соломона (Cauchy-Reed-Solomon). Иногда, для этих же целей используют [матрицу Вандермонда](#), и соответственно, метод носит название Вандермонда-Рида-Соломона (Vandermonde-Reed-Solomon).

Переходим к следующей проблеме. Для представления чисел в ЭВМ используется фиксированное число байтов. Соответственно, в наших алгоритмах мы не можем свободно оперировать произвольными рациональными, и тем более, вещественными числами. Мы просто не сможем реализовать такой алгоритм на ЭВМ. В нашем случае порождающая матрица состоит из целых чисел, но при обращении этой матрицы могут возникнуть рациональные числа, представлять которые в памяти ЭВМ проблематично. Вот мы и дошли до того места, когда на сцену выходят знаменитые поля Галуа.

Поля Галуа

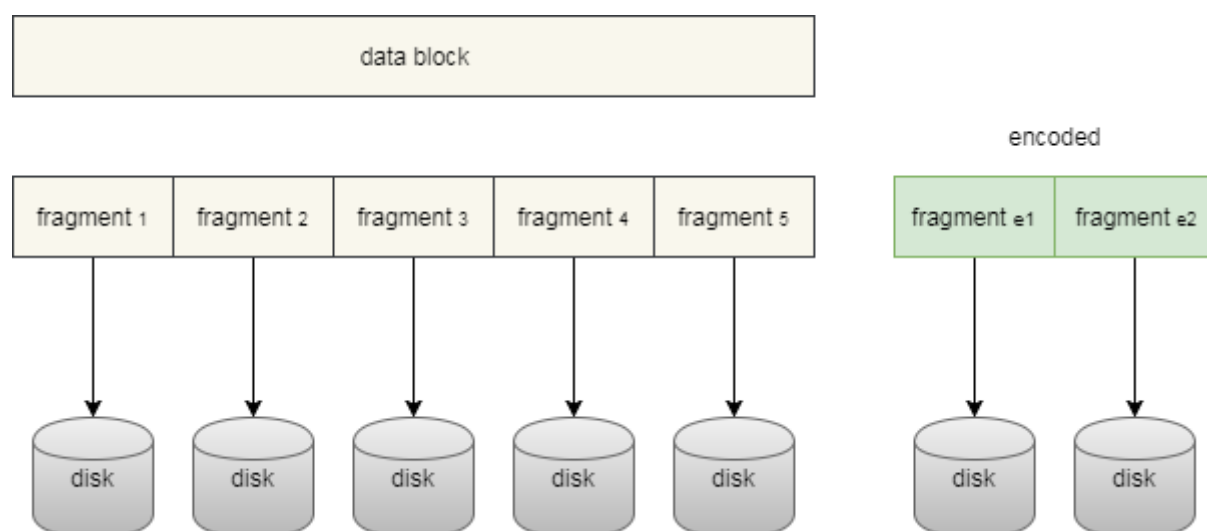
Итак, что такое поля Галуа? Начнём опять с поясняющих примеров. Мы все привыкли оперировать (складывать, вычитать, умножать, делить и прочее) с числами – натуральными, рациональными, вещественными. Однако, вместо привычных чисел, мы можем начать рассматривать произвольные множества объектов (конечные и/или бесконечные) и вводить на этих множествах операции, аналогичные сложению, вычитанию и т.д. Собственно говоря, математические структуры типа групп, колец, полей — это множества, на которых введены определенные операции, причем, результаты этих операций снова принадлежат исходному множеству. Например, на множестве натуральных чисел, мы можем ввести стандартные операции сложения, вычитания и умножения. Результатом опять будет натуральное число. А вот с делением все хуже, при делении двух натуральных чисел результат может быть дробным числом.

Поле – это множество, на котором заданы операции сложения, вычитания, умножения и деления. Пример — поле рациональных чисел (дробей). Поле Галуа — конечное поле (множество, содержащее конечное количество элементов). Обычно поля Галуа обозначаются как $GF(p)$, где p — количество элементов в поле. Разработаны методы построения полей необходимой размерности (если это возможно). Конечным результатом подобных построений обычно являются таблицы сложения и умножения, которые двум элементам поля ставят в соответствие третий элемент поля.

Может возникнуть вопрос – как мы всё это будем использовать? При реализации алгоритмов на ЭВМ удобно работать с байтами. Наш алгоритм может принимать на входе N байт исходных данных и вычислять по ним K байт избыточных. В одном байте может содержаться 256 различных значений, поэтому, мы можем создать поле $GF(2^8)$ и рассчитывать избыточные K байт, пользуясь арифметикой полей Галуа. Сам подход к кодированию/декодированию данных (построение порождающей матрицы, обращение матрицы, умножение матрицы на столбец) остаётся таким же, как и прежде.

Хорошо, мы в итоге научились по исходным N байтам конструировать дополнительные K байт, которые можно использовать при сбоях. Как это всё работает в реальных системах хранения? В реальных СХД обычно работают с блоками данных фиксированного размера (в разных системах этот размер варьируется от десятков мегабайт до гигабайтов). Этот фиксированный блок разбивается на N фрагментов и по ним конструируются дополнительные K фрагментов.

Процесс конструирования фрагментов происходит следующим образом. Берут по одному байту из каждого из N исходных фрагментов по смещению 0. По этим данным генерируется K дополнительных байтов, каждый из которых идет в соответствующие дополнительные фрагменты по смещению 0. Этот процесс повторяется для смещения – 1, 2, 3,... После того, как процедура кодирования закончена, фрагменты сохраняются на разные диски. Это можно изобразить следующим образом:



При выходе из строя одного или нескольких дисков, соответствующие потерянные фрагменты реконструируются и сохраняются на других дисках.

Теоретическая часть постепенно подходит к концу, будем надеяться, что базовый принцип кодирования и декодирования данных с использованием кодов Рида-Соломона теперь более или менее понятен. Если будет интерес к данной теме, то в следующих частях можно будет более подробно остановиться на арифметике полей Галуа, реализациях алгоритма кодирования/декодирования на конкретных аппаратных платформах, поговорить про техники оптимизации.

UPD. Вынесенный из [комментариев](#) список литературы по теме (спасибо [@whitedruid](#)):

- «Алгебраическая теория кодирования», Берлекэмп Э., 1971.
- «Теория кодов, исправляющих ошибки», Мак-Вильямс Ф., Слоэн Н.Дж., 1979.
- «Теория и практика кодов, контролирующих ошибки.», 1986.
- «Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение», Морелос-Сарагоса Р., 2005.
- «Кодирование с исправлением ошибок в системах цифровой связи», Кларк Дж., Кейн Дж., 1987.
- «Упаковки шаров, решетки и группы.», Конвей Дж.Н., Слоэн Н.Дж.А., 1990.

[@whitedruid](#): Книгу «Упаковка шаров, решет ки и группы» сам прочёл от носит ельно недавно — очень понравилось!

- «[Введение в алгебраические коды](#)» — лекции преподават еля МФТИ Юрия Львовича Сагаловича, оформленные в виде книг и.
- «[Коды Рида-Соломона с точки зрения обывателя](#)» — написано прост ым языком.
- «[Коды, исправляющие ошибки](#)» — крат ко, понят но и с карт инками :)
- «[Заметки по теории кодирования](#)», А. Ромащенко, А. Румянцев, А. Шень, 2011 год. — своего рода «шпаргалка» — крат ко, ёмко, однако т ребует некот орого уровня подгот овки.
- Также нельзя обойти стороной [семинары по «Теории кодирования»](#), которые регулярно организуются сотрудниками «Институт проблем передачи информации им. А.А. Харкевича Российской академии наук», г. Москва. Далее — по ссылкам — можно найти много интересного по темам, связанным с математической теорией кодирования, поднимаются актуальные вопросы в рамках дисциплины. [Например, про последовательное декодирование полярных и расширенных кодов Рида-Соломона](#). В поисковых же системах можно найти уже статьи автора.

Теги: [Коды Рида-Соломона](#), [поля Галуа](#), [erasure coding](#), [восстановление информации](#)

↑

+50

↓

🔖

232

👁

29,6k

💬

29

YADRO

51,00

Discover. Design. Develop.

👤

16,0

Карма

0,0

Рейтинг

10

Подписчики

@m-a-k-s-i-m

Пользователь

Комментарии 29

👤

mnuyam

24 августа 2017 в 10:10

🔖

🔖

↑

+1

↓

```
Pid_maxsim ! {message, self(), <<"Поля Галуа - хочу продолжения. ">>},
receive
  {Pid_maxsim, article, PoleGaluaPart2} -> read(PoleGaluaPart2)
end.
```

👤

m-a-k-s-i-m

24 августа 2017 в 10:41

🔖

🔖

🔖

🔖

↑

+7

↓

Была идея затронуть следующие темы:

- 1) Коды Рида-Соломона. Теория
- 2) Арифметика полей Галуа. Теория и реализация
- 3) Коды Рида-Соломона. Реализации
- 4) Коды Рида-Соломона. Программные оптимизации
- 5) Коды Рида-Соломона. Аппаратные оптимизации

Посмотрим, на что хватит времени...

https://habr.com/ru/company/yadro/blog/336286/

5/8

«Взялся за гуж, не говори, что не дюж!» С нетерпением жду продолжения))

 Nick_Sh1 25 августа 2017 в 05:13

#

🔖

🔗

🔄

↑ +1 ↓

Прочитав первое слово было подумал, что какого-то Максима хотите обозвать нехорошим словом :D
Аккуратнее надо...

 arandomic 24 августа 2017 в 10:11

#

🔖

↑ +3 ↓

«Вычеркнем из порождающей матрицы строки, соответствующие «пропавшим» данным.» — В реальной ситуации еще нужно выяснить какие данные пропали А и В или С и X0...

 Sirion 24 августа 2017 в 10:32

#

🔖

🔗

🔄

↑ 0 ↓

Если данные, соответствующие каждой буковке, пишутся на свой диск, то выяснение этого вопроса тривиально, нет?

 arandomic 24 августа 2017 в 12:05

#

🔖

🔗

🔄

↑ +1 ↓

Ну, алгоритм универсальный же.
Используется не только для восстановления данных в системах хранения информации, но и для коррекции ошибок при передаче данных.
А во входящем потоке у нас A-B-C-X0-X1 идут подряд. и тут либо добавлять еще избыточную информацию, чтобы разделить их, либо — поиск ошибок, Форни итп.

 AC130 24 августа 2017 в 19:27

#


🔖

🔗

🔄

↑ 0 ↓

Вообще там есть разделение на стирания (когда приёмник уверен что конкретный бит пропал при передаче) и ошибки (когда приёмник уверен что конкретный бит — вот такой, а на самом деле он был противоположный). Способность кода исправлять что-либо зависит от его расстояния. К примеру код Хэмминга (7, 4) имеет расстояние 3 и способен исправить один бит (при отсутствии стираний), либо два стирания (предполагая что ошибок нет). Если к нему добавить доп. бит контроля чётности, то расстояние увеличится до 4 и код будет способен исправлять одновременно одну ошибку и восстанавливать одно стирание (к примеру если пропал тот самый бит чётности, то код может исправить ошибку в первых 7 битах и восстановить стёртый бит чётности).

 m-a-k-s-i-m 24 августа 2017 в 10:35

#

🔖

🔗

🔄

↑ 0 ↓

Да, в реальных СХД хранится информация о том, какие фрагменты присутствуют на каждом диске. При выходе диска из строя, система составляет список потерянных фрагментов и передаёт эту информацию модулю восстановления данных.

 Sirion 24 августа 2017 в 10:24

#

🔖

↑ +3 ↓

Написано очень хорошо и понятно, спасибо. Продолжения жажду.

 sci_nov 24 августа 2017 в 10:35

#

🔖

↑ 0 ↓

Можете что-нибудь сказать о соответствии некоторой матрицы Коши порождающему полиному согласно определения кода Рида-Соломона? Насколько я помню, полиному соответствует некоторая матрица Вандермонда. Спасибо.

 starosta6123 24 августа 2017 в 12:33

#

🔖

↑ +1 ↓

Спасибо!
Отличное изложение, понял как 2*2.
Жду продолжения.

 whitedruid 24 августа 2017 в 12:38

#

🔖

↑ +5 ↓

Спасибо за статью!
В свою очередь хотел бы поделиться кратким списком литературы, а также парой ссылок — для тех, кто желает всесторонне изучить вопросы, связанные с алгебраической теорией кодирования.
Вероятно, книги уже неоднократно упоминались в статьях и комментариях по теме, однако, на мой взгляд, список литературы никогда не бывает избыточным :)

Из закладок

- «Алгебраическая теория кодирования», Берлекэмп Э., 1971.
 - «Теория кодов, исправляющих ошибки», Мак-Вильямс Ф., Слоэн Н.Дж., 1979.
 - «Теория и практика кодов, контролирующих ошибки.», 1986.
 - «Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение», Морелос-Сарагоса Р., 2005.
 - «Кодирование с исправлением ошибок в системах цифровой связи», Кларк Дж., Кейн Дж., 1987.
 - «Упаковки шаров, решетки и группы.», Конвей Дж.Н., Слоэн Н.Дж.А., 1990.
- Книгу «Упаковка шаров, решетки и группы» сам прочёл относительно недавно — очень понравилось!

- «Введение в алгебраические коды» — лекции преподавателя МФТИ Юрия Львовича Сагаловича, оформленные в виде книги.
- «Коды Рида-Соломона с точки зрения обывателя» — написано простым языком.
- «Коды, исправляющие ошибки» — кратко, понятно и с картинками :)

 SemenovNV 24 августа 2017 в 12:48

📌 ↕ ⬆

↑ +1 ↓

Спасибо за полезную подборку! Если не возражаете, можем добавить в конец самого поста (конечно, с упоминанием вас), чтобы больше людей увидели.

 whitedruid 24 августа 2017 в 13:22

📌 ↕ ⬆

↑ +1 ↓


Нет, не возражаю. Рад, что список литературы показался Вам полезным.
Добавьте, пожалуйста, ещё один документ: «Заметки по теории кодирования», А. Ромащенко, А. Румянцев, А. Шень, 2011 год. — своего рода «шпаргалка» — кратко, ёмко, однако требует некоторого уровня подготовки.
Также нельзя обойти стороной семинары по «Теории кодирования», которые регулярно организуются сотрудниками «Института проблем передачи информации им. А.А. Харкевича Российской академии наук», г. Москва. [Ссылка на math-net.ru](#) Далее — по ссылкам — можно найти много интересного по темам, связанным с математической теорией кодирования, поднимаются актуальные вопросы в рамках дисциплины. Например, про последовательное декодирование полярных и расширенных кодов Рида-Соломона. В поисковых же системах можно найти уже статьи автора.

 SemenovNV 24 августа 2017 в 15:26

📌 ↕ ⬆

↑ 0 ↓

Добавили в конец поста. Ещё раз спасибо за вклад!

 LorHobbit 25 августа 2017 в 10:04

📌 ↕ ⬆

↑ +1 ↓

У покойного Криса Касперски было довольно интересное введение в предмет — «Могущество кодов Рида-Соломона или информация, воскресшая из пепла» — как всегда у автора, статья спорная и отражающая частично его собственный опыт. На данный момент статья доступна в архиве, требует гуглокапчу: [archive.li/SBMti](#)

 SemenovNV 25 августа 2017 в 11:56

📌 ↕ ⬆

↑ 0 ↓

Спасибо за ссылку. Как и другие его статьи, читается очень интересно.

 Dmitry88 24 августа 2017 в 13:41

📌

↑ 0 ↓

Вы IBM?

 SemenovNV 24 августа 2017 в 14:23

📌 ↕ ⬆

↑ +1 ↓

Нет. И на всякий случай — мы не Рид и Соломон. Мы YADRO.

 Dmitry88 24 августа 2017 в 14:27

📌 ↕ ⬆

↑ 0 ↓

Это не звучало, как обвинение, скорее комплимент [oem.yadro.com](#)

Ничего зазорного в OEM нет.

 SemenovNV 24 августа 2017 в 14:39

📌 ↕ ⬆

↑ 0 ↓

Это уже обсуждалось в [комментах](#) к самой первой нашей статье.
Есть нормальный IT-бизнес, позволяющий финансировать свои разработки. Про эти разработки и пишем сюда в блог, почитайте наши статьи, вопросы к ним и ответы наших разработчиков — Хабр же технический ресурс, а не коммерческий.
Ну и на [yadro.com](#) про свои продукты пишем.

 Dmitry88 24 августа 2017 в 14:50

📌 ↕ ⬆

↑ 0 ↓

понял, удачи вам
А почему игнорируется AIX для VESNIN? Это технологическая проблема или политическая?

p.s. не мое дело, но yadro звучит отталкивающе, словно дитя Сколково

 SemenovNV 24 августа 2017 в 15:14

📌 ↕ ⬆

↑ 0 ↓

AIX требует подложки над аппаратурой в виде PowerVM, проприетарной технологии IBM, которая не доступна в OpenPOWER. Опять же, ввиду закрытости, мы не имеем возможности добавить в AIX поддержку работы на «Bare Metal». Проблема технологическая, решение которой, с политической точки зрения, никому не интересно.

 Dmitry88 24 августа 2017 в 15:17     0 



жаль, а то просадить по цене IBM не мешало бы в плане AIX серверов. Но их тоже можно понять, отдавать кусок рынка никому не хочется

 Gerrr 24 августа 2017 в 14:26   0 

Долгое время главным игроком в сфере безопасного хранения данных оставалась технология RAID.

Эту технологию сейчас кто-то смог обогнать по частоте применения? Интересно было бы прочитать кто. Даже если это произошло в какой-то конкретной сфере.






Очень буду ждать сравнения классических технологий RAID и вашей разработки. Хотя не до конца понимаю зачем свою реализацию противопоставлять самой технологии. Не правильнее ли запатентовать свой кастомный уровень.

 m-a-k-s-i-m 24 августа 2017 в 15:17     0 

По «частоте применения», возможно, технология RAID — лидер. По другим критериям, например, «объемы данных», картина, думаю, другая. Все известные мне «облачные» решения не используют RAID для хранения пользовательских данных. А объемы данных там гигантские. Современные классические СХД в связи с ростом объемов данных и увеличением требований к «физической» плотности систем, начинают сталкиваться с теми же ограничениями, что и «облачные системы» (где технологии RAID не находят массового применения). На самом деле, мы не противники RAID и несомненно есть и будут области, где RAID — оптимальный выбор. Мы рассматривали и такой вариант, но пришли к выводу, что использовать в нашем случае RAID — нецелесообразно.

 perhamm 24 августа 2017 в 15:23   0 

А можно Вас попросить рассказать каким образом при использовании erasure coding гарантируются различные схемы отказоустойчивости, например в документации к одному хранилищу данных написано, что поддерживаются схемы защиты 2d:1n или, например, 3d:1n_1d. То есть поддерживается выход из строя 2 дисков либо 1 ноды, а во втором случае поддерживает выход из строя 3 дисков либо 1 ноды и 1 диска. Вот как такие вещи реализуются? И означает ли это что 100% данные потеряются при выходе из строя, например, 3 дисков при схеме защиты 2d:1n? Но больше всего конечно интересен сам механизм.

 m-a-k-s-i-m 24 августа 2017 в 15:48     0 

Все зависит от архитектуры системы. В нашем случае, все «головы» кластера видят один и тот же набор дисков. Выход из строя «головы» не приводит к потере данных. Только сломавшийся диск требует запуска процедуры восстановления. Когда говорят про схему кодирования N+K (в технологиях erasure coding), подразумевается, что блок с данными разбивается на N фрагментов, по ним считаются дополнительные K, и все это «разлетается» по разным дискам. У нас схема по умолчанию 12+4, соответственно любые 4 диска можно потерять.

В других СХД, архитектура может быть другой. Конкретная «голова» может отвечать за какое-либо число дисков и выход из строя «головы» равнозначен поломке всех дисков, за которые она отвечает. Если, к примеру, «голова» отвечает за 3 диска, а схема 12+4, то можно потерять или 4 диска, или 1 диск и одну «голову» или ...