Курсовая работа

Алгориты детектирующие и исправляющие ошибки в системах хранения данных уровня RAID 6

Автор работы Солозобов Андрей Сергеевич

Группа №345 Мат-Мех СПбГУ

Руководитель Шевяков Михаил

Санкт-Петербург 2011

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СХД	Система хранения данных
КИО	Код, Исправляющий Ошибки
RAID	Redundant Array of Independent Disks

ВВЕДЕНИЕ

В системах хранения данных необходимо обеспечить сохранность, целостность информации и её восстановление после сбоев, случающихся в устройствах хранения. Для этого применяются различные способы избыточного кодирования данных.

При этом они все в той или иной степени решают задачи:

- исправления наибольшего числа ошибок
- введения в данные наименьшей избыточности
- достижения наименьшей сложности процессов кодирования и декодирования

При более близком рассмотрении становится понятно, что эти требования противоречивы и не могут быть выполнены одновременно. Именно поэтому существует большое количество кодов корректирующих ошибки, каждый из которых пригоден для решения своего круга задач. Среди них выделяют коды обнаруживающие ошибки и коды исправляющие ошибки. Эти два класса кодов тесно между собой связаны. В действительности, коды обнаружения ошибок принадлежат к тем же классам кодов, что и коды, исправляющие ошибки. И любой код, исправляющий ошибки, может быть также использован для обнаружения ошибок. При этом он будет способен обнаружить большее число ошибок, чем был способен исправить.

По способу работы с данными КИО делятся на блоковые, делящие информацию на фрагменты постоянной длины и обрабатывающие каждый из них в отдельности, и сверточные, работающие с данными как с непрерывным потоком.

1 КОДЫ, ИСПРАВЛЯЮЩИЕ ОШИБКИ В СХД

Типовым подходом для решения проблемы исправления ошибок в информации, связанных с помехами, является деление информации на блоки, называемые ***кадрами***, к каждому из которых применяется помехоустойчивое кодирование, позволяющее при необходимости обнаружить место ошибки и восстановить повреждённую информацию.

Если информацию делить на блоки по K бит, которые затем преобразуются в кодовые слова длины N бит (N>K). Здесь K называется ***размерностью кода*** (или ***числом информационных символов***), а N - ***длиной кода***. Такой способ кодирования, а, соответственно, и код, его реализующий, обозначается (N, K) и называется ***NK кодом***.

Если при этом криптограммы формируется таким образом, что исходной информации лишь добавляются проверочные символы, то такой код называется ***систематическим***, а иначе — ***несистематическим***.

Число R = K/N называется ***скоростью кода***. Чем ближе это число к единице, тем более быстрым и менее устойчивым становится код.

Простейшим способом защиты информации является её повторение (зеркалирование) — (N, 2N) код. Но, применительно к СХД, хоть он и позволяет аппаратно ускорить операцию считывания информации с носителей, зато он дорог в использовании (его скорость равняется одной второй) и защита от потери информации при сбое двух устройств СХД у него не стопроцентная (при выходе из строя двух зеркальных дисках информацию восстановить невозможно).

1.1 Линейные коды

Если при кодировании множество кодовых слов образует K-мерное линейное подпространство **C** N-мерного линейного пространства, то такой код называют ***линейным***. Будем рассматривать систематические линейные коды. Т.е. делается такое линейное преобразование исходных векторов, что размерность пространства при этом не меняется, и каждый вектор дополняется N-K координатами так, чтобы пространство не перестало быть линейным.

Задать блоковый код можно по-разному: линейной функцией, матрицей линейного преобразования, или же таблицей, где каждой совокупности из *k* информационных бит сопоставляется *n* бит кодового слова.

1.2 Линейные циклические коды

Код называется ***циклическим***, если он линеен и любой циклический сдвиг кодового слова также является кодовым словом.

Сопоставим каждому вектору (C_0, C_1, … , C_n-1) многочлен

$C(x)=C_0+C_1*x+ \dots +C_{n-1}x^{n-1}$

Пусть F – поле в алгебраическом смысле, тогда F[x] – множество многочленов от x с коэффициентами из F, являющееся коммутативным кольцом относительно сложения и относительно умножения. Теперь определим n-мерное векторное пространство над F:

$R_n=F[x]/ (x^n-1)$, его элементами как раз и являются всевозможные C(x). Умножение C(x) на x по модулю x^n-1 соответствует циклическому сдвигу.

1.3 Порождающие полиномы

Идеалом C кольца R_n называется такое линейное его подпространство, что $C(x)\in C$ => для $\forall$ $R(x)\in R_n$, верно $R(x)*C(x)\in C$.

Г**лавным идеалом** называют множество всевозможных произведений элементов из R_n на некоторый фиксированный многочлен, называемый **порождающим**.

Теорема:

1. В R_n есть единственный нормированный многочлен g(x) наименьшей степени
2. В пространстве R_n каждый идеал является главным
3. Каждый циклический код имеет порождающий многочлен g(x)
4. Все кодовые слова каждого конкретного циклического кода кратны своему порождающему полиному и могут быть записаны в виде произведения g(x)*f(x), где $f(x)\in F[x]$ и deg f(x) < n – deg g(x)
5. Сам порождающий многочлен g(x) является делителем x^n-1.

(Доказательство этой теоремы есть в книге №2 из списка использованной литературы)

Таким образом циклический код в кольце F[x] состоит из произведений g(x) на многочлены степени меньше n-r, где r=deg g(x). Получается, что есть всего n-r линейно независимых таких произведений {g(x), x*g(x), …, x^{n-r-1}*g(x)} и размерность кода равна n-r.

1.4 Коды Рида — Соломона

Эти коды были изобретены в 1960 году сотрудниками лаборатории Линкольна Массачуссетского технологического института Ирвином Ридом и Густавом Соломоном. Первое применение код Рида — Соломона получил в 1982 году в серийном выпуске компакт-дисков. Сейчас они используются в различных областях хранения и передачи данных: CD, DVD, Blu-ray дисках, DSL, WiMAX, сотовых каналах связи и др.

2 КОДЫ, ИСПРАВЛЯЮЩИЕ ОШИБКИ В СХД УРОВНЯ RAID 6

Массивы типа RAID 1, 2, 3, 4, 5 защищают СХД от единичного сбоя. Но в момент, когда система восстанавливается из предыдущего сбоя, она задействует все диски на протяжении всего процесса восстановления. В такой ситуации вероятность дополнительного сбоя возрастает, что в случае указанных типов RAID будет фатально. Так же с течением времени ёмкость жестких дисков постоянно растёт, а вероятность сбоя при работе с ячейкой памяти остаётся практически неизменной, и в результате эффективность защиты такого RAID массива оказывается недостаточной.

RAID 6 – это общее название RAID массивов, позволяющих защитить СХД от сбоя двух устройств хранения и при этом требующих дополнительно ещё два устройства хранения.

2.1 RAID6 на основе кода Рида — Соломона

Для реализации этого вида RAID 6 требуется посчитать два контрольных синдрома. Первый синдром получается как сумма многочленов в стрипе. Для получения второго синдрома производится суммирование многочленов стрипа, домноженных каждый на x^i, где i - номер диска, которому соответствует блок в стрипе.

Если сбои произошли в a-ом и b-ом информационных блоках, то при исправлении ошибок приходится решать систему уравнений вида:

$$\begin{cases} B_a + B_b = P \\ B_a \cdot x^a + B_b \cdot x^b = Q \end{cases}$$

Система 1

где B_k – k-тый блок данных в страйпе, а P и Q – полученные синдромы чётности и Рида-Соломона соответственно.

Она разрешима только при обратимости элементов x^i и x^j (решается, выражением D_i из нижнего уравнения, D_j из верхнего и его подстановкой в нижнее). Именно поэтому порождающий полином должен быть неприводим в кольце многочленов с двоичными коэффициентами.

2.2 RAID DP

В аббревиатуре RAID DP, DP расшифровывается как Double Parity – двойная чётность.

Пусть у нас есть СХД, состоящая из M+2 физических дисков, каждый из которых содержит в себе N блоков данных. Теперь нарисуем матрицу размера Nx(M+2), которая для определённости будет представлять нашу СХД.

В своей работе этот тип массива использует подсчёт двух независимых синдромов чётности – обычного (суммирование блоков, лежащих в одной строке), который используется в RAID 5, и диагонального.

Диагональная чётность с номером i получается суммированием блоков с индексами

[k, ((k+i-2) mod M) + 1], при k ∈ [1:M]. Для наглядности можно взглянуть на картинку №1 из приложения А.

При сбое одного устройства хранения, восстановление происходит с применением обычной горизонтальной чётности, и при этом диагональная чётность просто вводит дополнительный уровень защиты.

Если же выходит из строя второе устройство, то целиком задействуется механизм Double Parity. Пусть для определённости отказали в работе диски с индексами i и j и значения блоков, хранившихся на них, теперь нам неизвестны (рис. №1 приложения А). Тогда:

1. По построению, L-тая диагональная чётность с индексом L=((i+1) mod M)+1 имеет только один неизвестный блок [x1,y1], который мы восстанавливаем (рис. №2 приложения А). Дальше восстанавливаем, единственный оставшийся неизвестным блок [x1,j], лежащий в x1-ой строке (рис. №3 приложения А). Потом восстановим блок в диагонали, которая проходит через [x1,j] (рис. 4 приложения А)
2. Потом аналогичная последовательность операций проводится начиная с T-ой диагонали, где T=((j+1) mod M)+1.

3 АЛГОРИТМИЧЕСКИЕ ОПТИМИЗАЦИИ

На этапе проектирования алгоритма очень важным является создание максимально линейного кода программы, не содержащего каких-либо ветвлений. Это даёт возможность оптимально использовать конвейер команд процессора.

Для ускорения операции нахождения остатка от деления многочлена делитель должен иметь старший член 128 или 64 степени, а остальные члены должны быть не старше 63 степени. Это позволяет проводить факторизацию, используя два умножения и два сложения многочленов.

При решении Системы 1 получается формула:

$$\begin{cases} B_a = P + B_b \\ B_b = \dfrac{P \cdot x^{-a} + Q}{x^{b-a} + 1} \end{cases}$$

При вычислении значений блоков потерянных данных используется довольно медленная операция деления многочленов. Так как при инициализации системы мы уже знаем максимальное количество дисков в ней, то будет разумно предподсчитать все возможные знаменатели этой формулы. Поиск обратного многочлена рационально проводить алгоритмом Евклида.

3.1 Команды intrinsics и работа с кэшем

В современных процессорах существует многоуровневая система кэшей, которые заполняются из памяти целыми кэш-линиями. Таким образом, если мы начали восстанавливать систему после сбоя одного или двух дисков, то при проходе по страйпу мы на самом деле получим в кэше данные сразу из нескольких страйпов за счёт того, что размер кэш-линии в несколько раз больше, чем размер нашего блока, и у современных процессоров составляет 64 байта (в четыре раза больше размера одного блока в 128 бит). Но чтобы этот процесс работал, блоки должны быть выровнены на границу 64 байт и кратны размеру кэш-

линии, иначе они будут кэшироваться кусками, что приведёт к снижению производительности.

Также современные процессоры поддерживают массу расширений набора команд, в частности, наборы SSE, SSE2, SSE4 и CLMUL. Эти команды очень удобны в использовании и при написании кода на языке C вызываются так называемыми intrinsics командами, которые, как утверждает компания Intel, позволяют производить дополнительные оптимизации на этапе компиляции кода программы.

Мы выяснили, что при восстановлении одного или нескольких дисков, когда уже произведён проход по одному страйпу и запрошены данные с соответствующих дисков, мы можем сравнительно быстро провести обработку ещё нескольких страйпов, оказавшихся в кэше. Так как мы знаем, какие страйпы собираемся обрабатывать дальше, то за время текущей обработки хорошо было бы подгрузить из памяти в кэш необходимые данные следующих страйпов. Для этого есть специальная команда _mm_prefetch из набора SSE, которая позволяет подгрузить одну кэш-линию начиная с указанного адреса в ближайший к процессору кэш. Причём помимо адреса у этой команды есть ещё один параметр, указывающий, какого рода операции мы намерены дальше совершать с этими данными. В частности, от него зависит, будут ли эти данные при вытеснении из кэша помещены в более дальний кэш или сразу в память, не «пачкая» его (не вытесняя из него уже лежащие там актуальные данные).

Когда же мы занимаемся восстановлением системы, то при записи восстановленных по синдромам данных на диск они будут кэшироваться, тем самым «пачкая» кэш. Понятно, что пользователю данные одного или двух дисков вряд ли нужны и поэтому кэшировать их не стоит. В расширении SSE есть команды _mm_stream_pi и _mm_stream_ps, которые позволяют записывать данные «мимо» процессорного кэша. Понятно, что не стоит применять эти команды при пересчёте синдромов страйпа, связанном с изменением данных. Синдромы часто используемых страйпов как раз стоит кэшировать. И здесь уже встаёт решаемый эмпирически очень важный вопрос выбора оптимального количества дисков в системе и размера дискового страйпа, чтобы его данные и синдромы хорошо ложились в кэш.

3.2 Умножение многочленов

Очень важным оказывается использование команды PCLMULQDQ из набора CLMUL, которая позволяет перемножать 64-битные половины 128-битных переменных типа _m128i (именно поэтому удобно выбирать соответствующие многочлены факторизации и размеры дисковых блоков). При этом умножение в процессоре проходит с выключенными цепями переноса, т.е. при переполнении разряда он сбрасывается на ноль без увеличения ближайшего старшего разряда. Эта команда полностью соответствует описанному в нашем поле умножению и тем самым позволяет гораздо быстрее его выполнять.

4 ДОСТИЖЕНИЯ В РАМКАХ КУРСОВОЙ РАБОТЫ

В рамках курсовой работы:

- Были изучены основные алгоритмы программной организации RAID 6 массивов в СХД.
- Был написан тестирующий модуль, проводящий проверку корректности работы тестируемого алгоритма и замеры его производительности на выбранных наборах тестов.

  Тестировалось три алгоритма RAID 6. Первый, написанный на языке Ассемблера, был взят из ядра СХД Avrora компании AvroRAID. Второй, написанный на языке C, был получен из рук генерального директора компании AvroRAID. Работа первых двух алгоритмов была основана на использовании фактор-многочлена 128 степени.
- Были изучены наборы процессорных расширений MMU, XMM, SSE, SSE2, SSE4a, SSE4b, CLMUL и их intrinsics аналоги в языке C, а также различные способы реализации алгоритмов полиномиального кодирования.
- На языке C была написана библиотека подпрограмм для вычислений в поле многочленов с фактор-многочленом $\mathbf{x^{64} + x^4 + x^3 + x + 1}$ с использованием intrinsics команд.
- Был написана третья реализация алгоритма RAID 6 на основе вышеупомянутого полинома 64-ой степени, использующая большинство вышеупомянутых оптимизаций.

Наборы тестовых параметров системы включали в себя декартово произведение количеств физических дисков в системе от 8 до 32 с шагом в 2 диска и размеров дисковых страйпов от 4 Кбайт до 128 Кбайт с шагом в 4 Кбайта.

В результате сравнения производительности трёх вышеперечисленных алгоритмов оказалось, что третий алгоритм, основанный на полиноме 64-ой степени работает в среднем в 1,2 — 1,5 раза медленнее, чем алгоритм, основанный на полиноме 128-ой степени,

который в свою очередь оказался в 3 раза более медленным, чем алгоритм из ядра СХД Avrora, написанный на языке Ассемблера.

5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фёдоров А.Р. ''Задача восстановления утерянных дисков в массиве с использованием арифметики конечных колец''
2. Ф.Дж. Мак-Вильямс, Н.Дж.А. Слоэн ''Теория кодов, исправляющих ошибки'' Bell Laboratories 1977г
3. В.В. Лидовский '''Теория информации'' Спутник 2004г
4. У. Питерсон, Э. Уэлдон ''Коды, исправляющие ошибки'' Мир 1976г
5. Intel® 64 and IA-32 Architectures Software Developer's Manual

http://www.intel.com/products/processor/manuals/

6. Intel® 64 and IA-32 Architectures Optimization Reference Manual

http://www.intel.com/products/processor/manuals/

7. Intel® Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode

http://www.intel.com/products/processor/manuals/

8. Intel® 64 and IA-32 Architectures Application Note TLBs, Paging-Structure Caches, and Their Invalidation

http://www.intel.com/products/processor/manuals/

9. Porting and Optimizing Multimedia Codecs for AMD64 architecture on Microsoft® Windows®

http://download.microsoft.com/download/5/b/5/5b5bec17-ea71-4653-9539-204a672f11cf/MMCodec_amd64.doc

10. http://www.bytemag.ru/articles/detail.php?ID=6702
11. http://www.insidepro.com/kk/027/027r.shtml

ПРИЛОЖЕНИЕ А

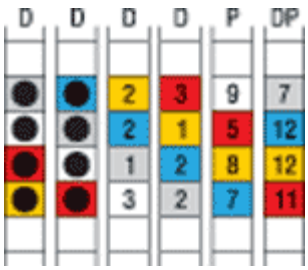

рис. №1

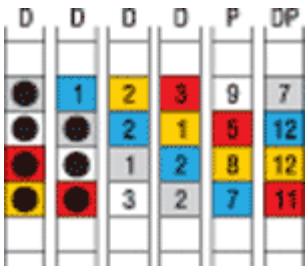

рис. №2

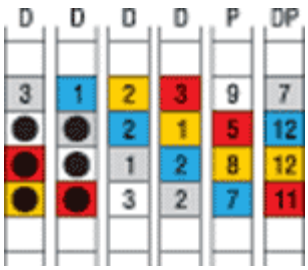

рис. №3

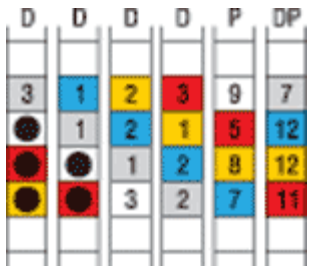

рис. №4