Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Алгоритмы расчета RAID 6

Курсовая работа студента 345 группы
Калмука Александра Игоревича

Научный руководитель ..... *Короткевич А.И., AvroRAID*

Санкт-Петербург
2012

Содержание

Введение

Системы хранения данных (СХД) являются специализированными средствами для хранения и передачи информации. Для этих целей используются различные спецификации по количеству дисков, размещению данных и контрольных сумм.

«Аппаратные» СХД (ASICs) – это готовые решения с использованием специализированных аппаратных компонентов, разработанных производителем специально для этих изделий. Данные решения представляют собой законченные решения и за редким исключениям из компоненты не могут быть использованы в других СХД. Так, большинство ведущих производителей «аппаратных» СХД (например HP, IBM, EMC) поставляют для своих СХД, в том числе, и специализированные жесткие диски. При этом использование дисков сторонуих производителей недопустимо, и приводит к снятию изделия с гарантийной поддержки.

«Программные» СХД - отличаются тем, что используют компоненты общего назначения, которые могут быть использованы в любых других решениях, включая серверы или даже персональные компьютеры. Все специфические для СХД функции в них реализованы на программном уровне. По сути «программные» СХД представляют собой специализированные серверы, выполняющие роль системы хранения данных.

Преимущество «программных» СХД перед «аппаратными» заключается в том, что для построения СХД не требуются никакие специализированные компоненты, а комплектующие общего назначения x64 стремительно снижаются в цене и, на сегодняшний день, успешно могут конкурировать по производительности со специализированными «аппаратными».

Ключевой технологией для обеспечения надежности хранения данных на дисковых массивах является возможность восстановления данных при выходе из строя одного (или нескольких) жестких дисков. Для этого используются различные методы коррекции кода – так называемые избыточные дисковые массивы RAID (Redundant Array of Independent Disks). Восстановление происходит за счет вычисления и хранения некоторой избыточной информации (контрольных сумм или синдромов), которая требует дополнительного дискового пространства.

Оптимальным с точки зрения надежности и экономии дискового пространства является алгоритм RAID6 (с чередованием и двумя независимыми «чётностями» блоков). Данный алгоритм позволяет восстанавливать данные при выходе из строя до двух жестких дисков в дисковой группе.

Целью моей работы было исследование различных алгоритмов расчета RAID 6, реализация своего алгоритма в виде библиотеки функций, представляющих операции в RAID (см. «операция» в п. «Определения»), сравнение и измерение производительности этих алгоритмов. Стоит отметить, что моя курсовая является частью коллективной работы.

Определения

Блок - наибольший фрагмент логического адресного пространства, принадлежащий одной дорожке, и физически расположенный на одном носителе памяти СХД.

Дорожка - фрагмент логического адресного пространства, принадлежащий одному страйпу, размер которого делит размер страйпа.

Операция - одна из пяти основных функций, предоставляемых алгоритмами уровня RAID6: генерация синдромов P и Q; пересчёт синдромов P и Q при изменении данных на одном из дисков; восстановление одного диска данных по синдрому P; восстановление одного диска данных по синдрому Q; восстановление двух дисков с данными по P и Q.

Поле Галуа - конечное поле GF(q) из q элементов. Известно, что поле GF(q) существует тогда и только тогда, когда $q=p^n$, где p – простое. При этом все поля из q элементов изоморфны между собой.

Система хранения данных (СХД) - комплексное программно-аппаратное решение по организации надёжного хранения информационных ресурсов и предоставления гарантированного доступа к ним.

Синдром - синдром, вычисляемый как сумма (XOR) исходных данных. Термин «четность» связан с правилом вычисления операции XOR, когда бит значения равен единице, если среди битов операндов нечетное число единиц, и нулю в противном случае.

Страйп - наименьшая единица логического адресного пространства СХД, для которой при записи происходит пересчёт синдромов.

Алгоритм

Я использовал метод расчёта RAID 6, основанный на подсчёте синдрома чётности и полиномиального синдрома *Рида-Соломона*.

Код Рида-Соломона имеет вид:

$$Q(g0, \ldots, gn)=xn*gn + xn-1* gn-1+ \ldots+g0$$

где gi – блоки данных фиксированного размера, интерпретируемые как элементы векторного пространства, а элементы {xi} лежат в некотором поле.

В алгоритме расчета RAID 6, описанном в [3], используется представление поля Галуа GF(2^8) как остатков вычетов по модулю неприводимого многочлена, то есть однобайтовых слов, с конструктивно описанным умножением. Я использовал представление поля Галуа GF(2^8) в матричной алгебре [5]. Таким образом, вычислительная сложность операций в RAID сводится к сложности операции умножения матрицы на вектор. Поэтому основной задачей стала быстрая реализация этой операции для произвольных матриц, а также для матриц специального вида.

Реализация

Так как нашей задачей является максимально ускорить вычисления, то было решено писать код на языке ассемблера, что позволило более эффективно использовать регистры, чем после трансляции из языка Си.

Предполагается, что мы получаем из некоторой внешней функции страйп. Далее происходит обработка страйпа по дорожкам, то есть к каждой дорожке данных применяются операции RAID 6. Рассмотрим подробнее, что происходит внутри. Каждая операция представляет собой отдельную функцию, вычислительная сложность которой определяется количеством операций «умножение матрицы на дорожку».

Дорожка имеет размер 64 байта и разделяется на 4 поддорожки размером 16 байт каждая. Каждый такой блок данных помещается на регистр xmm (как раз 16 байт) и интерпретируется как вектор из восьми координат, каждая координата таким образом занимает 2 байта на регистре.

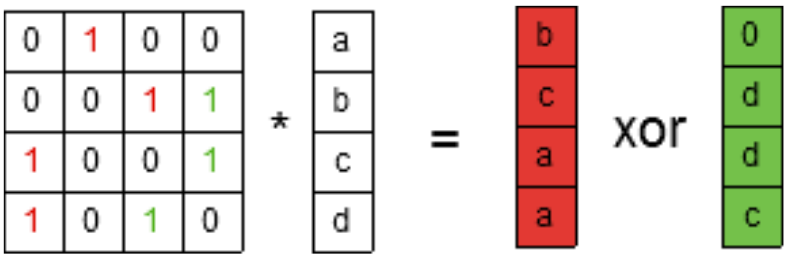
Для обеспечения корректности работы с полиномиальным кодом Рида-Соломона необходимо, чтобы используемое множество матриц с традиционными операциями умножения, сложения, и умножения на скаляр образовывало поле. Это условие является достаточным, как можно понять из [5]. Не исключено, что требование можно ослабить, однако изучения этого вопроса не потребовалось. После изучения статьи [5] стало ясно, что в качестве такого поля подходит поле Галуа из 256 = 2^8 матриц, в котором образующим мультипликативной группы является матрица Фробениуса над GF(2). Поле GF(2) было выбрано как самое простое для реализации операций умножения и сложения.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Итак, дорожка загружена на 4 регистра (на самом деле это всего лишь порядка одного чтения из памяти) . Так как поле конечно и состоит из 256

элементов, то я реализовал 256 функций умножения для каждой конкретной матрицы на вектор. Потребовалось по некоторому алгоритму сгенерировать все эти функции, при условии что количество операций в среднем на одну такую функцию должно быть как можно меньше.

Идея в том, что умножение матрицы на вектор можно представить как умножение этой матрицы, разложенной в сумму, на этот вектор. При хотелось, чтобы умножение каждого слагаемого на вектор было очень быстрым. Будем раскладывать матрицу в сумму следующим образом. Пока возможно перебрасываем из каждой строки матрицы по одной «1» в новую матрицу. Таким образом, получаем матрицу содержащую 8 единиц. Когда какая-нибудь из строк обнуляется, перестаем использовать ее, но по-прежнему выделяем из оставшихся строк по одной «1» формируем новую матрицу. Если выписать эти матрицы в порядке получения, то легко понять, что у них количество единиц убывает. Далее, инструкция PSHUFB расширения SSE3 позволяет перемешивать байты на регистре. Потому умножение на каждую такую матрицу это одно применение инструкции PSHUFB к исходному вектору (собственно для этого в такую сумму матрица и была разложена). Это изображено схематически на рисунке для матрицы 4x4:



По такому алгоритму были сгенерированы функции умножения на все матрицы, кроме M, так как работа с ней занимает большую часть времени работы операций RAID.

Для матрицы M я применил отдельную оптимизацию. Можно заметить, что если транспонировать ее, то умножение на вектор сводится к одному PSHUFB (для зеленых «1») и сдвигу (для красных «1»).

0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0

Небольшое число регистров xmm объясняет, почему нельзя выбрать размер обрабатываемой дорожки слишком большим, так как в противном случае данные перестанут помещаться на регистры и число обращений к памяти возрастет. Количество же матриц в поле ограничено снизу максимальным числом дисков (я считал, что это 128, и таким образом, поле матриц 4x4 из 16 элементов не подошло). При этом, чем меньше размерность матрицы, тем меньше процессорных инструкций требуется на умножение при описанном выше подходе. Эти два факта обосновывают то, что размерность 8x8 не случайна.

Тестовый полигон

Использовался тестовый полигон, разработанный в рамках такого же проекта в 2010/2011 годах.
Для оценки эффективности реализации сравнивалась производительность пяти основных функций, предоставляемых алгоритмами уровня RAID6:

- генерация синдромов P и Q;
- пересчёт синдромов P и Q при изменении данных на одном из дисков;
- восстановление одного диска данных по синдрому P;
- восстановление одного диска данных по синдрому Q;
- восстановление двух дисков с данными по P и Q.

Алгоритм проверялся на корректность операций восстановления и пересчета.

Для примера ниже приведен алгоритм проверки корректности восстановления данных про утрате двух дисков:

1. Случайным образом выбираются два различных номера дисков N1 и N2, чей выход из строя будет эмулироваться.
2. Данные с блоков N1 и N2 сохраняются в отдельных массивах D1 и D2 соответственно.
3. Блоки N1 и N2 инициализируются случайными данными.
4. Вызывается функция восстановления СХД по обоим синдромам P и Q.
5. Проверяется совпадение данных из восстановленных блоков N1 и N2 и данных, сохранённых в массивах D1 и D2.

Системные характеристики

Тестирование реализаций алгоритмов проводилось на выделенном сервере под управлением операционной системы Scientific Linux 6.1. (Red Hat 4.4.4-13) На момент проведения тестирования на машине параллельно с тестирующей программой выполнялись лишь сервисные приложения операционной системы.

```
$ uname -a
Linux 123 2.6.32-71.el6.x86_64 #1 SMP Wed Sep 1 01:33:01 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux
$ cat /proc/version
Linux version 2.6.32-71.el6.x86_64 (mockbuild@x86-007.build.bos.redhat.com)
```

(gcc version 4.4.4 20100726 (Red Hat 4.4.4-13) (GCC)) #1 SMP Wed Sep 1 01:33:01 EDT 2010

Замеры производительности

Тесты позволяют получать среднее время выполнения в процессорных тиках, а также их математическое ожидание и дисперсию, что позволяет статистически оценить насколько устойчив алгоритм к случайным событиям восстановления данных на разных машинах.

Процедуре замера скорости при запуске передаются три параметра:

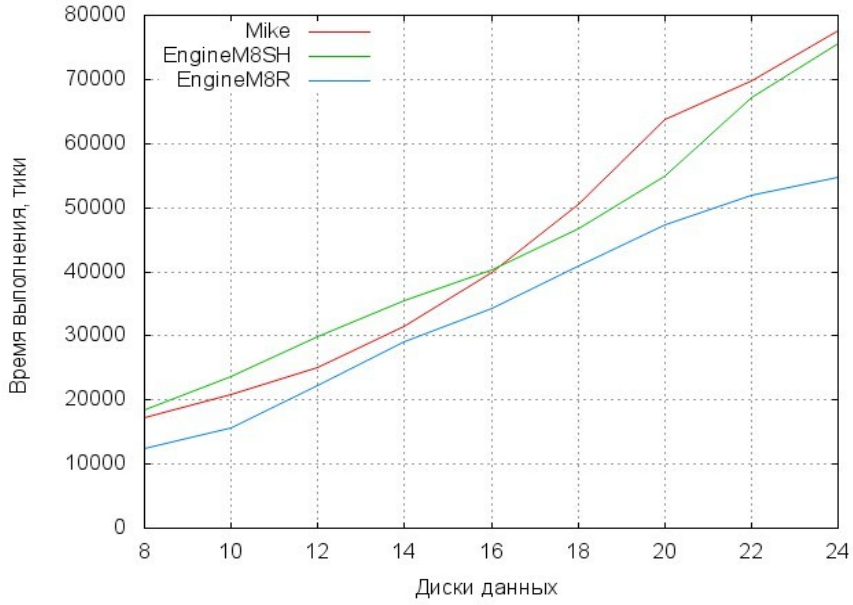
- количество жестких дисков в эмулируемой СХД (включая служебные диски, предназначенные для хранения синдромов);
- размер одного диска эмулируемой СХД («ширина» страйпа)
- Число элементарных тестов

Элементарные тесты выполняются заданное число раз и для полученных массивов времён исполнения операций считаются значения среднего времени исполнения E для каждой операции и среднеквадратичного отклонения времени исполнения операции от посчитанного среднего времени исполнения D. Среднее время считается следующим образом: все результаты для одной операции сортируются по возрастанию, далее от полученного отсортированного массива «отщепляется» какая-то часть самых больших и самых маленьких по величине результатов (в проведенных тестах использовалось значение 10, то есть 1/10 верхних и нижних значений откидывались). Проводилось это для того, чтобы предотвратить влияние разнообразных скачков показаний, которые могут быть следствием малопредсказуемых вещей вроде запуска какой-нибудь системной службы или падения напряжения.

Результаты измерений

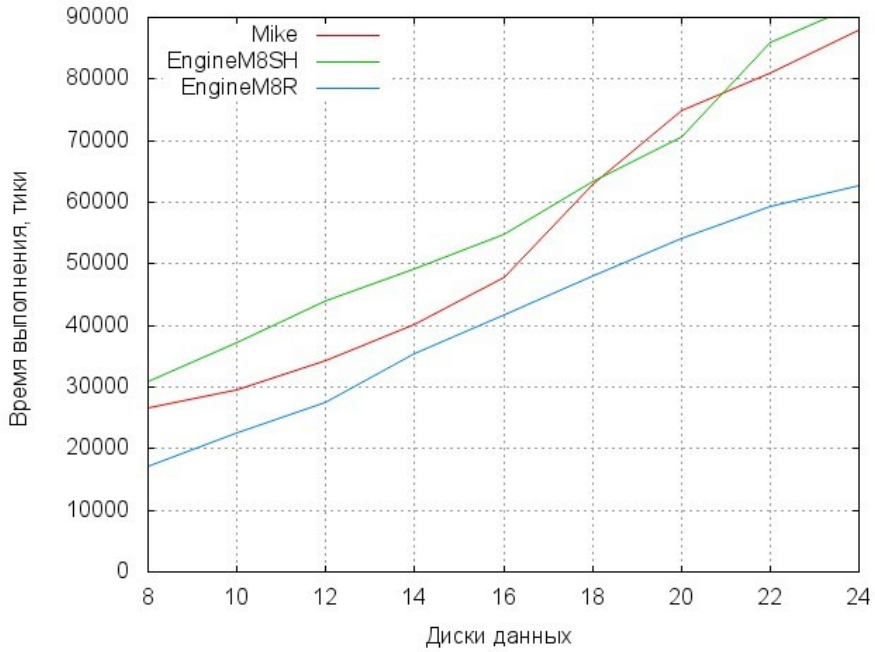
Ниже приведены результаты сравнения алгоритма с алгоритмом используемым в *СХД Avrora* — продукте разработанным компанией *AvroRAID,* а также с еще одной реализацией алгоритм RAID 6 в рамках проекта этого года.

Генерация синдромов P и Q.



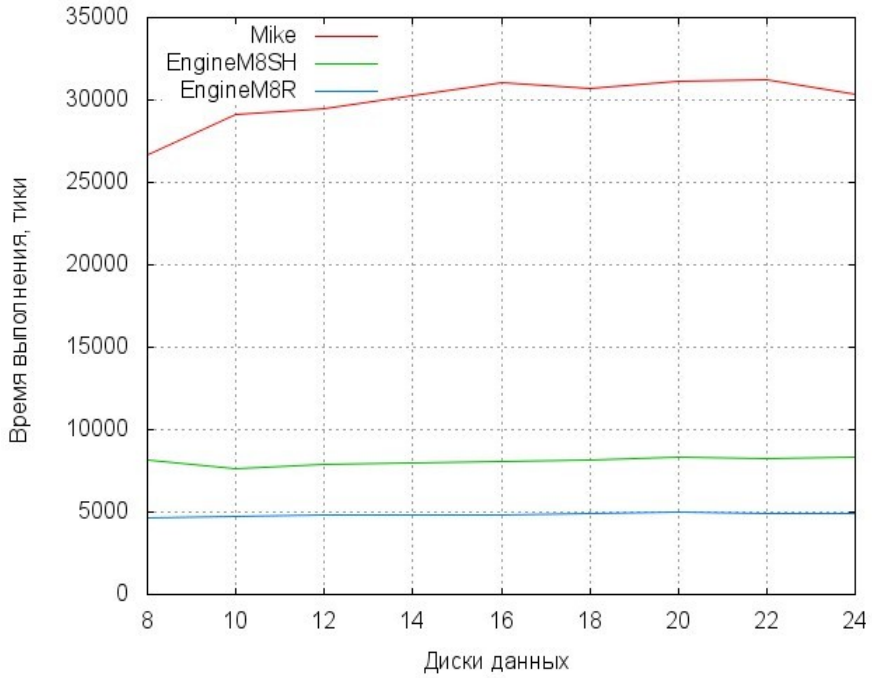
Диски	Алгоритм Майка	EngineM8R	EngineM8SH
8	17151	18413	12430
10	20755	23696	15612
12	25097	29942	22355
14	31420	35401	29060
16	39908	40205	34198
18	50545	46717	40869
20	63778	54930	47226
22	69806	67195	51946
24	77523	75622	54646

Восстановление двух дисков с данными по P и Q.



Диски	Алгоритм Майка	EngineM8R	EngineM8SH
8	26625	30862	17099
10	29654	37114	22556
12	34313	44059	27615
14	40071	49211	35343
16	47777	54807	41738
18	63036	63350	48135
20	74996	70666	54041
22	80891	85838	59254
24	87917	92246	62637

Пересчет синдромов P и Q при изменении данных на одном их дисков



Диски	Алгоритм Майка	EngineM8R	EngineM8SH
8	26646	8148	4667
10	29096	7626	4719
12	29493	7853	4804
14	30270	7982	4867
16	31042	8059	4847
18	30684	8148	4925
20	31159	8329	4959
22	31209	8279	4931
24	30345	8327	4942

Выводы

Мной была реализована библиотека функций, выполняющих все требуемые операции, и протестирована на нескольких процессорах. Как видно, мой алгоритм с учетом погрешностей сопоставим с алгоритмом Майка, однако проигрывает второму алгоритму, который использует 128-байтную дорожку, а не 64-байтную, но при этом используя большее число обращений к памяти.

Используемые источники

1. System V Application Binary Interface AMD64 Architecture Processor Supplement (Draft Version 0.99.5)
2. Утешев А.Ю. Математика отказоустойчивых дисковых массивов http://pmpu.ru/vf4/codes/raid
3. H. Peter Anvin (2006-2011), The mathematics of RAID-6, http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf
4. Федоров А.Р., Задача восстановления утерянных дисков в массиве с использованием арифметики конечных колец
5. Утешев А.Ю. Поля Галуа http://pmpu.ru/vf4/gruppe/galois