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

Алгоритмы расчёта RAID 6

Курсовая работа студента 345 группы

Тюшева Кирилла Игоревича

Научный руководитель …….. Короткевич А.И.

Санкт-Петербург
2012

Содержание

1. 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. Спецификация RAID 6 дополнительно хранит два диска с контрольными суммами и может восстанавливать до двух дисков с данными.

Целью курсовой работы являлось исследование различных алгоритмов расчёта RAID 6, реализация своего алгоритма в виде библиотеки функций, сравнение и измерение производительности этих алгоритмов.

Стоит отметить, что моя курсовая является частью коллективной работы.

2. Определения

Блок - наибольший фрагмент логического адресного пространства, принадлежащий одному страйпу, 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 - вычисляемые избыточные данные массивов RAID-5 и RAID-6.

Синдром четности - 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.

Тик процессора - единица измерения внутреннего времени системы.

3. Алгоритм

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

Код Рида-Соломона имеет вид:

$$Q(g_0, \dots, g_n) = x^n * g_n + x^{n-1} * g_{n-1} + \dots + g_0$$

где g_i – блоки данных фиксированного размера, интерпретируемые как элементы векторного пространства, а элементы $\{x^i\}_{i=0}^n$ 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, т.е. однобайтовых слов, с конструктивно описанным умножением. Я использовал представление поля Галуа $GF(2^8)$ в матричной алгебре [4].

Таким образом, вычислительная сложность алгоритма сводится к операции умножения матрицы размера 8x8 на вектор. Поэтому основной задачей стала быстрая реализация этой операции для произвольных матриц, а также для матриц специального вида. В частности особенно важно как можно быстрее умножать на матрицу, в которую переходит X, чтобы быстро вычислять код Рида – Соломона по схеме Горнера. В соответствии с [4] в качестве X была взята матрица вида:

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

4.Реализация

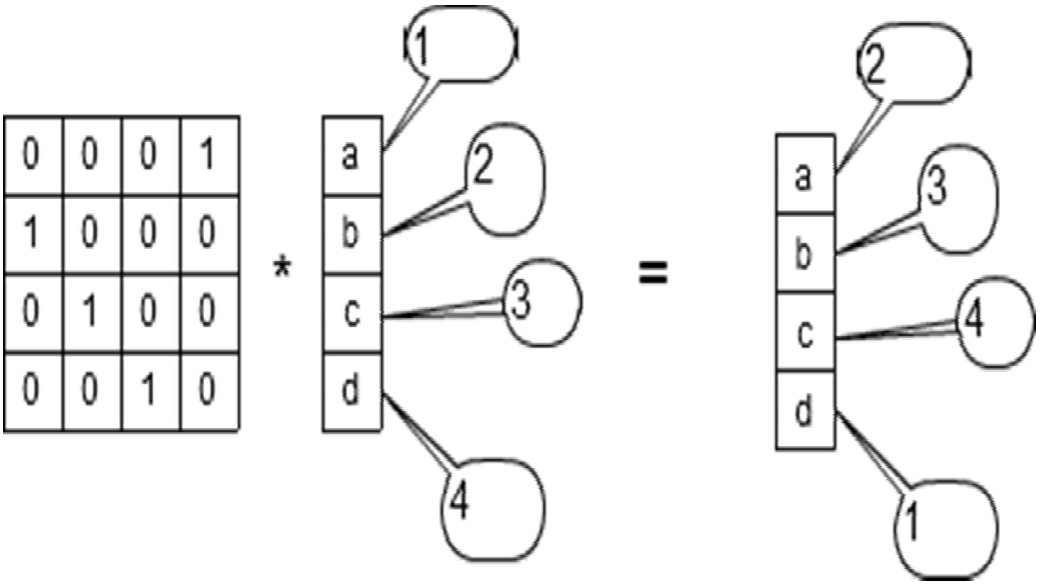
Так как нашей задачей является максимально ускорить вычисления, то было решено писать код на языке ассемблера, что позволило более эффективно использовать регистры, чем после трансляции из языка Си.

Предполагается, что мы получаем из некоторой внешней функции дорожку. Далее к этой дорожке применяется одна из операций. Каждую операцию выполняет отдельная функция.

Размер дорожки может быть разным. При маленьком размере дорожки нам хватает регистров для любых промежуточных вычислений без обращения к памяти, но за один проход по дискам мы обрабатываем небольшой объём данных. При большом размере дорожки за один проход по дискам мы обрабатываем большой объём данных, но нам приходиться больше обращаться к памяти.

Размер дорожки в моей реализации равен 128 байт. Причём каждая координата вектора хранится на отдельном регистре XMM.

При таком подходе любую перестановку координат вектора можно производить "в уме", что позволяет сократить число операций процессора. То есть, в дальнейших вычислениях можно считать, что соответствующие координатам вектора регистры теперь содержат другие координаты вектора:



Можно заметить, что матрицу X, определённую выше, можно представить как сумму двух матриц специального вида. Первая матрица делает циклический сдвиг координат вектора. Вторая содержит три единицы в первой строчке, а в остальных позициях нули. Циклический сдвиг можно не делать. То есть надо сделать только три сложения соответствующие трём единицам второй матрицы. Таким образом, умножение на X выглядит так:

```
pxor   %xmm3,      %xmm7
pxor   %xmm4,      %xmm7
pxor   %xmm5,      %xmm7
```

Здесь регистры XMM0, XMM1, … , XMM7 содержат координаты вектора. То есть вектор – это (XMM0, XMM1, … , XMM7).

Следующее умножение на X будет выглядеть по-другому, так как теперь вектор – это (XMM7, XMM0, … , XMM6):

```
pxor   %xmm2,      %xmm6
pxor   %xmm3,      %xmm6
pxor   %xmm4,      %xmm6
```

Через 8 умножений на X код станет таким же, как в первом варианте.

Обращения к памяти происходят группами данных, кратными размеру линии кэша процессора.

Для регистров XMM используются только команды MOV и XOR, что позволяет использовать эту реализацию на большом числе процессоров**.**

5.Тестовый полигон

Использовался тестовый полигон, разработанный в рамках курсовых работ 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.

Все алгоритмы проверялись на корректность операций восстановления и пересчета.

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

Программные компоненты

Проверка корректности

Элементарной проверки корректности является тест, которому при запуске передаются два параметра:

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

Тесты корректности выполнялись для всего диапазона количества дисков, участвующих в измерениях, для 8Kb ширины страйпа.

Замеры производительности

Элементарной единицей замера производительности является тест скорости, запускающий по очереди каждую из пяти операций и измеряющий количество тиков, которое прошло за время её выполнения, т.е. разность между значениями в процессорном счётчике тиков непосредственно перед запуском операции и после её завершения. Текущее состояние счётчика процессорных тиков получалось командой RDTSC().

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.

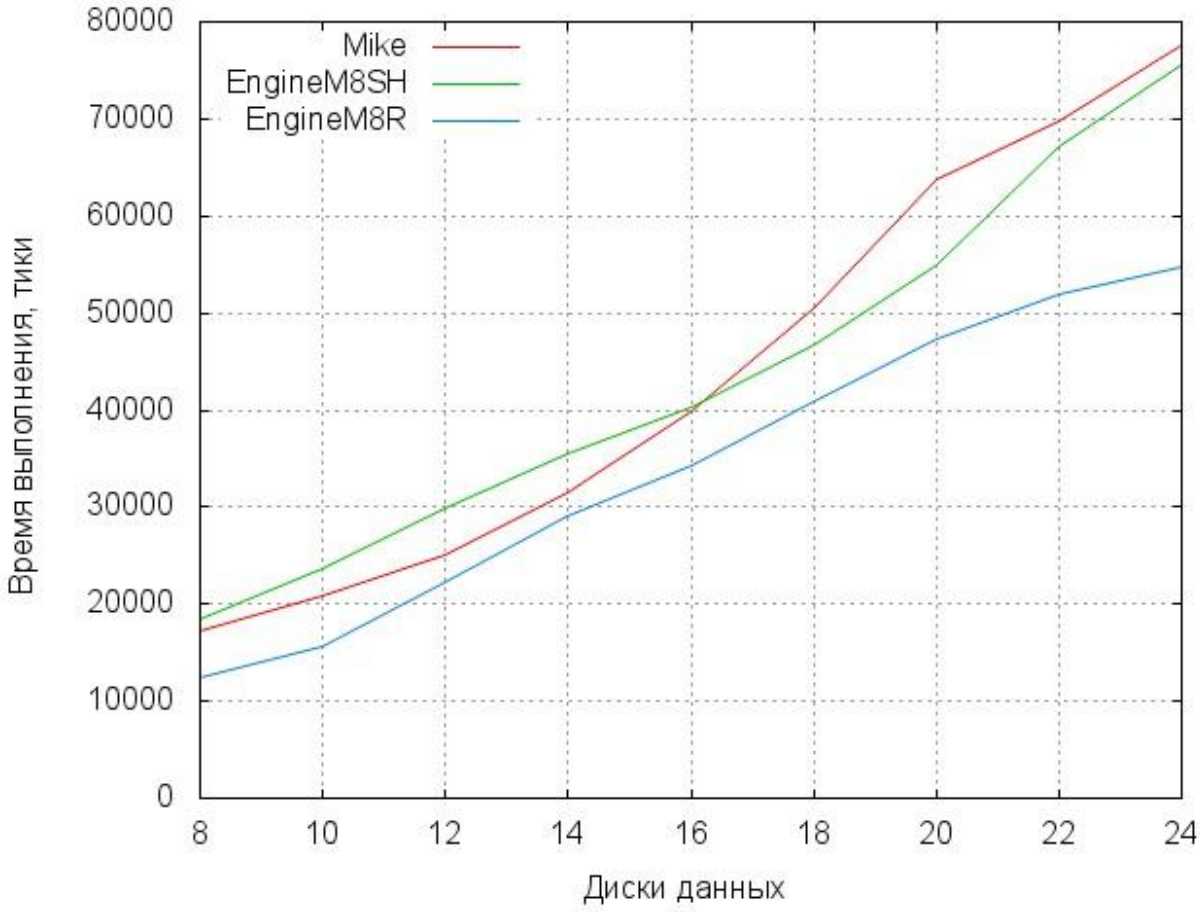
Замеры производительности выполнялись для 8Kb ширины страйпа.

6. Результаты измерений

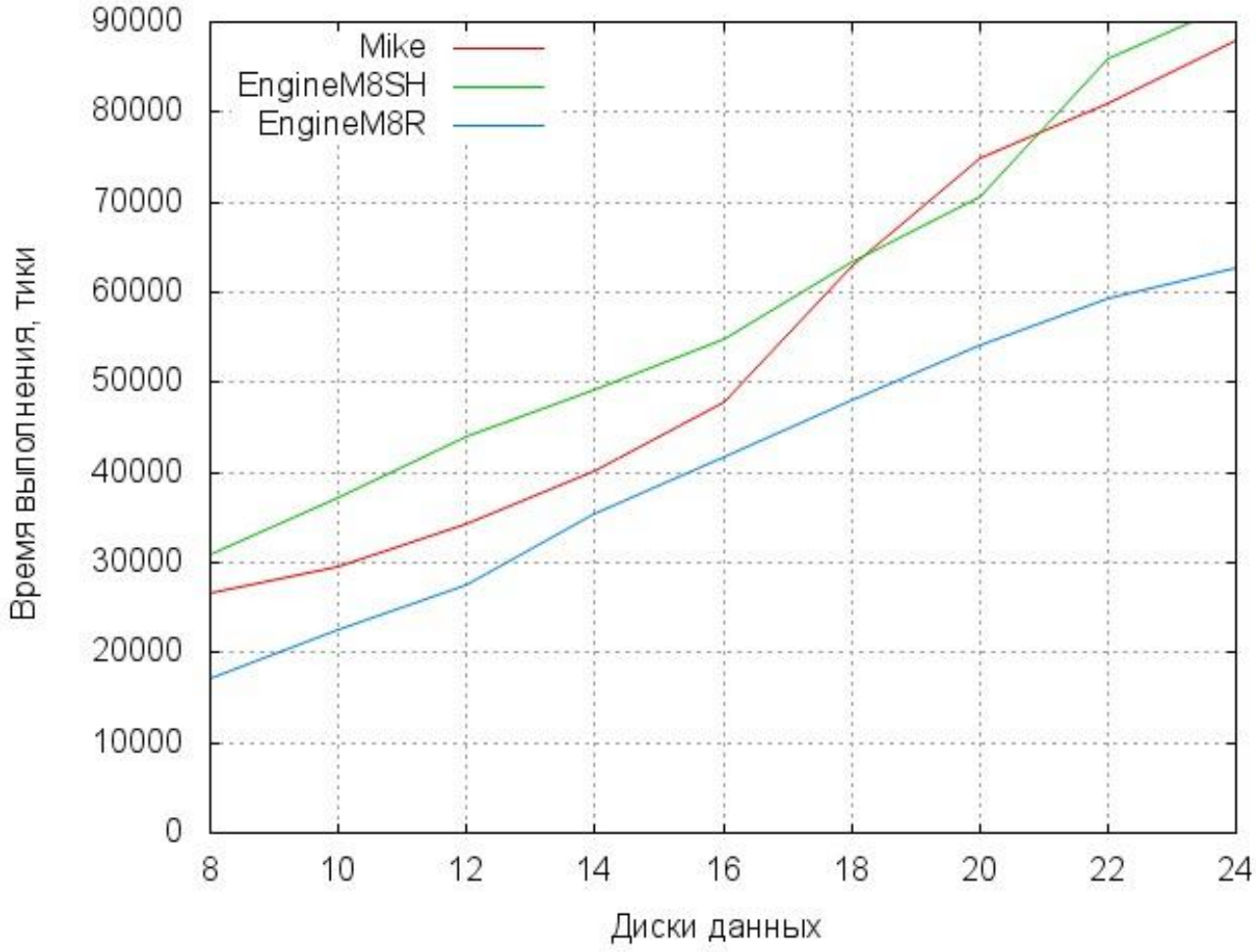
Ниже приведены результаты сравнения следующих алгоритмов:

1. EngineM8R: Описанная выше реализация
2. Mike: Алгоритм, используемый в СХД AVRORA, продукте, разработанном компанией AvroRAID
3. EngineM8SH: Ещё одна реализация описанного выше алгоритма, написанная в рамках проекта этого года

Генерация синдромов P и Q



Восстановление двух дисков с данными по P и Q



Выводы

Как видно, при выполнении всех операций лучший результат показал алгоритм EngineM8R. Стоит однако отметить, что преимущество над алгоритмом Майка оказалось не столь большим, по сравнению с количеством выполняемых операций, что, по-видимому объясняется менее оптимальной работой с памятью, так как ввиду нехватки регистров, каждое значение читается два раза, один раз для синдрома P, и один раз для синдрома Q. Так же отметим, что попытки оптимизации EngineM8R путем дополнительного раскрытия цикла по числу дисков не принесли ощутимого выигрыша в скорости, разница между реализациями попадала в погрешность измерения.

Данные измерений

Генерация синдромов			
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 2х дисков			
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

7.Заключение

В ходе работы над курсовой были выполнены следующие задачи:

1. Изучены основные алгоритмы программной реализации RAID 6 массивов.

2. Для описанного выше алгоритма реализован генератор кода библиотеки функций расчёта RAID 6 для любого количества дисков при размере вектора 128 байт.

3. Проведены измерения производительности различных реализаций алгоритмов расчёта RAID 6 и сделаны выводы по их результатам.

8. Список литературы

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. Утешев А.Ю. Поля Галуа http://pmpu.ru/vf4/gruppe/galois
5. Федоров А.Р., Задача восстановления утерянных дисков в массиве с использованием арифметики конечных колец