Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Система хранения данных. Поддержка избыточного кодирования. Оптимизация, настройка и аппробация выбранного алгоритма под поставленную задачу. Оценка полученных результатов.

Курсовая работа студента 345 группы
Мальчевского Михаила Андреевича

Научный руководитель ……. Косякин А. Н.
ведущий разработчик, ООО "Артек"

Санкт-Петербург
2010

Содержание

Введение

Во многих организациях главным активом является информация, поэтому нужно обеспечивать надежное хранение информационных ресурсов и быстрый доступ к ним.

В настоящее время повышенный интерес вызывает распределенное хранение данных, обладающее рядом преимуществ перед традиционным прямым подключением дисковых массивов к серверам: сеть хранения данных представляет собой архитектурное решение для подключения внешних устройств хранения данных, таких как дисковые массивы к серверам таким образом, чтобы операционная система распознала подключённые ресурсы как локальные.

Основные преимущества:

- Высокая масштабируемость;
- Высокая производительность и надежность;
- Простота администрирования;
- Эффективное восстановление работоспособности после сбоя;

На данный момент существует множество решений, но на небольших предприятиях они являются редкостью из-за слишком высокой стоимости. Целью проекта является создание надежной сети хранения данных, работающей на дешевом, доступном оборудовании и обладающей всеми вышеуказанными свойствами.

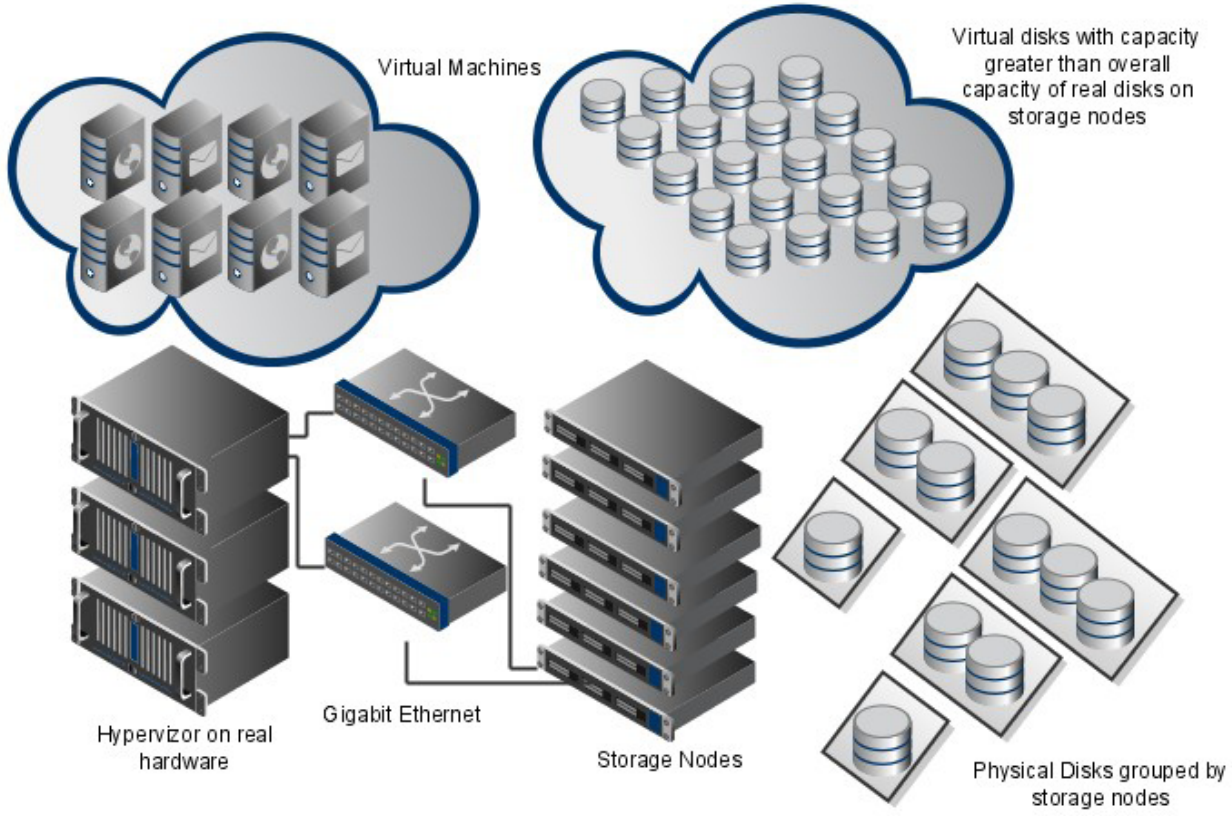


Рис.1 Распределенное хранилище

Cirrostratus является сетью хранения данных, оптимизированной под «облачные вычисленея». Поверх некоторого числа реальных серверов, имитируется в несколько раз большее число виртуальных серверов, которые внешне ничем не отличаются от реальных. Открытым стоит вопрос, как недорого имитировать жесткие диски для этих виртуальных машин (сегодня виртуальный сервер работает на сервере 1, завтра - на сервере 2; не переставлять же диски руками!), причем при сохранении приемлемой производительности и высокой надежности, ведь современные винчестеры довольно ненадежны и могут выйти из строя, либо быть бракованными. Поэтому предлагается использовать избыточное кодирование, чтобы защитить данные от отказов узлов хранения, но при этом эффективно использовать имеющееся дисковое пространство и обеспечивать приемлемую нагрузку на CPU.

Алгоритмы избыточного кодирования.

В данной курсовой рассматривается вопрос подбора оптимальных параметров для алгоритма хранения, выбранного заранее. Описание алгоритмов будет приведено, впрочем, в более общей форме, потому что подавляющее большинство алгоритмов устроены схожим образом. В проекте «Cirrostratus» используется Minimal Density Code для сценария RAID-6, но также это мог бы быть RDP, EVENODD и некоторые другие.

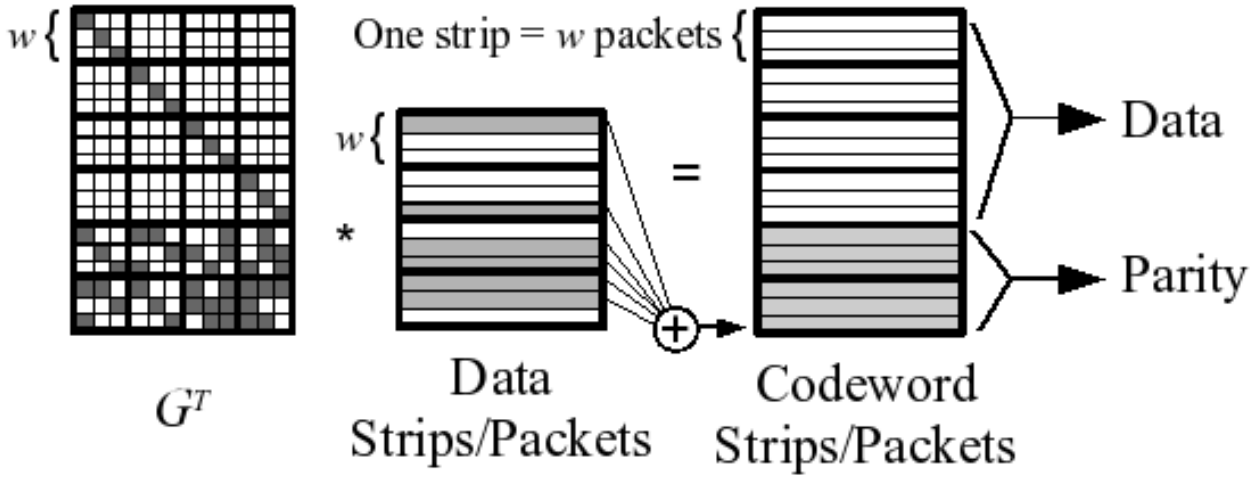
Параметры алгоритмов.

Чтобы описать параметры, которые возникают в ходе применения алгоритмов избыточного кодирования, необходимо объяснить, как кодируются данные в этих алгоритмах.

Представим, что нужно закодировать большой файл, который можно представить в виде k частей. Файл, который получится на выходе, будет содержать n = k + m частей. При этом мы рассматриваем алгоритмы, в которых система устойчива к потере m любых частей. То есть, потеряв любые m частей выходного файла, можно, с помощью оставшихся k частей, полностью восстановить потери. Итого, мы получаем первые два параметра — k и m.

Далее, каждая часть файла(назовем их блоками) делится на s полос. А каждая полоса, соответственно, разбита на w частей, каждая размером PS байт. Таким образом, получили еще три параметра, которые влияют на производительность системы в целом.

Процесс кодирования происходит следующим образом. Кодирование происходит для всех полос отдельно, одна за другой.



Каждая полоса умножается на кодирующую матрицу, состоящую из единиц и нулей. Причем единицы в данном случае показывают строчки, к которым применяется побитовая операция XOR. Таким образом, каждая сточка выходной полосы получается применением операции XOR к некоторым строкам входной.

Чаще всего, верхняя часть кодирующей матрицы является единичной матрицей, потому что первые k частей полосы остаются без изменений, и только последние m «собираются» по-другому.

Определение оптимальных параметров.

Размер буфера данных.

Размер буфера данных — размер полосы, которая кодируется на отдельном шаге. Первоначальное тестирование с различными значениями этой величины показало, что производительность системы очень сильно зависит от этого параметра. Но при этом было заметно, что время, затрачиваемое на то, чтобы закодировать один и тот же файл могло меняться на очень большую величину. В среднем разброс был на 25%. Тогда возникла идея, что такой колоссальный разброс может быть связан с операциями записи на диск и чтения с него. В программу были внесены необходимые изменения. Вместо чтения с жесткого диска и записи на него, она стала генерировать случайную последовательность байт, производить над ней действия по кодированию и заполнять нулями выбранное место в буфере(оперативной памяти).

Данное усовершенствование позволило определить роль данного параметра гораздо более точно. Во-первых, разброс времени на кодирование тестов одного и того же размера перестал превосходить 5%. Во-вторых, стало видно, что данная величина фактически не оказывает влияния на систему в целом. А значит она может быть выбрана такой, как будет удобно. Далее, во всех тестах эта величина была примерно равна 100 килобайтам. Так как данная величина зависит от четырех параметров: k, m, w и PS, то она не может быть выбрана точно. Так что она находилась в окрестности этого значения. 100 килобайт — количество, достаточно большое, чтобы операции чтения/записи были сравнительно эффективными, но, с другой стороны, не настолько большое, чтобы забить всю оперативную память сервера, так как в реальном случае на сервере может одновременно выполняться большое количество задач.

На данный момент отказ от записи на диск - специально сделанное допущение, ибо учет этого параметра требует более детальной проработки (необходимо дополнительно учесть среднее время случайного поиска данных, размер дискового кеша и т.д.).

Параметр w.

Данный параметр отвечает за то, сколько строчек будет в каждой из частей полосы. Он отличается от других параметров тем, что зависит от конкретного алгоритма кодирования. Для каждого алгоритма известно, какое конкретно w стоит выбирать для получения оптимальной производительности. Для алгоритма RDP, который был нами реализован исключительно в целях сравнения его производительности с другими, например, оптимальное w должно равняться k, то есть количеству частей, на которые изначально разбивается кодируе-

мый файл. Для алгоритма же MD, наоборот, чем больше w, тем лучше. И оптимальная производительность в данном случае достигается при w, стремящемся к бесконечности.

PS(Packet Size).

Еще один очень важный параметр, сильно влияющий на производительность алгоритмов кодирования. Первоначально, тестирующая программа для выбранного алгоритма, k, m и w полностью тестировала кодирование со всеми вариантами PS от 4 до 10000 байт. Уже тогда по полученным результатам можно было сказать, что производительность в целом растет с ростом PS. Хотя, конечно, зависимость не была строго монотонной.

Зависимость можно описать следующим образом. Производительность росла с ростом PS до некоторой точки «максимума». Эта точка глобально максимальной производительности появлялась в тот момент, когда алгоритм наилучшим образом использовал кэш L1. К сожалению, кривая зависимости ни монотонно возрастала, ни убывала от «пика». Хуже того, были «впадины» между соседними близкими значениями из-за коллизий между вхождениями в кэш.

Но полный расчет продолжался слишком долго, а нужно было сделать это для ряда алгоритмов, при этом меняя k и m. Необходимо было найти хорошее упрощение этой тестовой программы.

Тогда программа перестала искать глобально лучшее значение. Вместо этого был применен новый подход, который работает таким образом. Тестируется *регион* r значений PS, путем проверки всех значений от r до r + 36(PS должен быть кратным числа 4 из-за устройства машинной памяти и размера переменных в битах). *Производительность региона* — среднее арифметическое этих пяти производительностей.

Поиск оптимума начинался с того, что тестировались все регионы, которые являлись степенями двойки от 64 до 32К. Среди них выбирался наилучший регион, а также его «правый» и «левый» соседи(регионы, который в два раза больше и в два раза меньше, соответственно). Затем, между этими регионами посередине измеряли значения еще двух регионов, для всех пяти выбирали оптимальный и процесс повторялся. Это происходило до тех пор, пока не оставался всего один регион, который и являлся искомым. В среднем, данный алгоритм хуже, чем алгоритм полного перебора на 1-2%, но обеспечивает огромный выигрыш по времени по сравнению с последним.

Вполне ожидаемо также, что с ростом k, m, w оптимальный размер PS уменьшается.

Оптимизация алгоритмов.

Помимо манипуляций с параметрами для алгоритмов, возможны оптимизации, которые модифицируют работу самих алгоритмов. Здесь я рассмотрю два варианта подобных оптимизаций, которые так или иначе используются в проекте.

Оптимизация количества операций XOR

Первое, что приходит в голову, когда пытаешься придумать, как можно ускорить работу алгоритма кодирования — это каким-то образом уменьшить количество операций над словами из битов. Наметки по тому, как это сделать, придумать несложно — если мы два раза считаем сумму(последовательность операций XOR) нескольких слагаемых, и, при этом, каждая из сумм содержит некую общую «подсумму», то можно сначала вычислить ее, а потом, с помощью данного «предвычисления», считать основные суммы. Нетрудно видеть, что общее количество операций при таком подходе сократиться. И это только самый простой вариант.

В общем случае данную задачу(оптимизации) можно записать в терминах графов и она оказывается NP-полной. Но ведь нам и не нужно каждый раз решать ее. Достаточно вместе с алгоритмом хранить и оптимальную последовательность вычислений, по которой будет происходить кодирование.

С декодированием все становится несколько сложнее. Для небольших m и k мы можем хранить такие последовательности вычислений для всех вариантов. Для остальных случаев нужно придумывать отдельные хитрости. К счастью, остальные случаи пока не входят в нашу проект, поэтому мы можем обойтись и минимальным набором трюков для оптимизации.

Данная техника была использована практически повсеместно как во время выбора из множества алгоритмов лучшего, так и во время подбора оптимальных параметров для выбранного.

Оптимизация последовательности операций XOR

Вторая мысль, уже не настолько лежащая на поверхности, которая приходит на ум во время попыток оптимизации — не менять операции XOR, а попробовать переставлять их. Это нужно для того, чтобы постараться использовать возможности кэша компьютера наилучшим образом. В зависимости от порядка, в котором будут происходить операции, будут также меняться и данные, которые временно хранятся в кэше.

Не буду сильно углубляться в эту оптимизацию, скажу только, что правильное использование этой техники, несмотря на кажущуюся ее неэффективность, позволяет добиться существенного роста производительности для практически любого алгоритма.

Заключение.

В данной работе был проведён анализ параметров, присущих каждому алгоритму избыточного кодирования. Были показаны способы нахождения оптимальных значений. Также были рассмотреты варианты увеличения производительности, касающиеся реализации выбранных алгоритмов.

Список литературы

[1] BLAUM, M., BRADY, J., BRUCK, J., AND MENON, J. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. IEEE Transactions on Computing 44, 2 (February 1995), 192– 202.

[2] BLAUM, M., AND ROTH, R. M. On lowest density MDS codes. IEEE Transactions on Information Theory 45, 1 (January 1999), 46–59.

[3] BLOMER, J., KALFANE, M., KARPINSKI, M., KARP, R., LUBY, M., AND ZUCKERMAN, D. An XOR-based erasure-resilient coding scheme. Tech. Rep. TR-95-048, International Computer Science Institute, August 1995.

[4] CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row diagonal parity for double disk failure correction. In 3rd Usenix Conference on File and Storage Technologies (San Francisco, CA, March 2004).

[5] J. L. Hafner, V. Deenadhayalan, K. K. Rao and A. Tomlin, "Matrix Methods for Lost Data Reconstruction in Erasure Codes," FAST-2005: 4th Usenix Conference on File and Storage Technologies, San Francisco, December, 2005, pp. 183-196.

[6] C. Huang, J. Li and M. Chen, "On Optimizing XOR-Based Codes for Fault-Tolerant Storage Applications," ITW'07, Information Theory Workshop, IEEE, Tahoe City, CA, September, 2007, pp. 218-223.

[7] J. Luo, L. Xu and J. S. Plank, "An Efficient XOR-Scheduling Algorithm for Erasure Code Encoding," Technical Report Computer Science, Wayne State University, December, 2008.