Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра информационно-аналитических систем

Гусева Мария Максимовна

Исследование возможности сжатия хеш-значений при дедупликации данных

Бакалаврская работа

Допущена к защите
И. о. заведующего кафедрой:
к. ф.-м. н., доцент Е. Г. Михайлова

Научный руководитель:
старший преподаватель Д. В. Луцив

Рецензент:
к. ф.-м. н., доцент Е. Г. Михайлова

Санкт-Петербург
2016

Saint-Petersburg State University
Department of Analytical Information Systems

Guseva Maria

Investigation of hash value compression in data deduplication

Bachelor's Thesis

Admitted for defence
Acting Head of Department:
Associate Professor E. G. Mikhailova

Scientific Supervisor:
D. V. Luciv

Reviewer:
Associate Professor E. G. Mikhailova

Saint-Petersburg
2016

Оглавление

Введение

Регулярное резервное копирование является обычной практикой для защиты от аппаратных сбоев и ошибок пользователей. Сжатие данных перед передачей может помочь существенно увеличить пропускную способность[1]. Действенным методом для эффективного уменьшения передаваемых данных является дедупликация[2].

Дедупликация – это метод сжатия массива данных, при котором находятся и удаляются дубликаты данных. Сначала данные разбиваются на блоки, по которым считаются хеш-значения, а затем эти хеш-значения сравниваются, и только при совпадении уже сравниваются сами блоки данных.

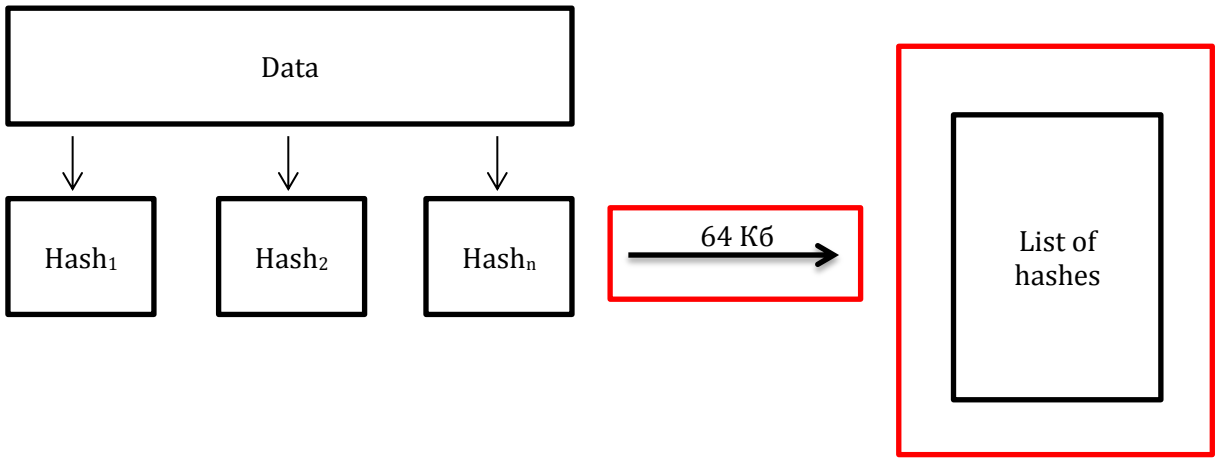
В процессе дедупликации генерируется большой массив хеш-значений. Например, для блоков по 4 Кб на 4 Тб данных потребуется 8 Гб памяти только для хеш-значений (если брать алгоритм, в котором 1 хеш занимает 8 байт). Кроме того, требуется память для хранения идентификаторов блоков и других метаданных.

Одной из стадий дедупликации является сравнение полученных хеш-значений с теми, которые уже хранятся в базе данных. Для ускорения процесса сравнения можно хранить список хешей в in-memory базе данных. Но такой способ подходит не для всех хранилищ, т.к. размер in-memory баз данных намного меньше обычных. Поэтому для расширения границ ее использования можно применить сжатие к хеш-значениям перед хранением.

1. Постановка задачи

Цель данной работы – создание структуры данных для дедупликации для получения существенной экономии памяти, выделенной для хранения метаданных.

Рисунок 1. Компрессия блока хеш-значений в процессе дедупликации.



Описание задачи:

На вход подаются хеш-значения, полученные из данных (одно хеш-значение на каждые 4 Кб данных). После того, как набирается 64 Кб хеш-значений, требуется произвести компрессию. То есть компрессия производится не над одним хеш-значением, а над блоком размера 64 Кб, состоящим из хеш-значений.

В памяти будет храниться какое-то количество блоков сжатых хеш-значений разного размера (т.к. не все блоки одинаково поддаются сжатию). Хеш-таблица – это структура данных, обладающая свойством амортизированной сложности доступа, которая достигается при надлежащем подборе параметров (то есть в среднем поиск элемента имеет временную сложность. O(1))[3]. Соответственно, хотелось бы сохранить это свойство даже после сжатия, чтобы не приходилось разжимать всю хеш-таблицу на стадии сравнения хеш-значений во время дедупликации. То есть требуется разжимать только тот сжатый блок, где, вероятно, лежит такое же хеш-значение, а не все блоки.

Таким образом, требуется найти оптимальный способ хранения хеш-значений, а также наиболее подходящий по времени работы и степени сжатия алгоритм. Кроме того, необходимо произвести предварительное исследование возможности сжатия обычного списка хеш-значений (грубая оценка степени сжатия).

2. Предварительная работа

Перед решением поставленной задачи необходимо выяснить есть ли вообще смысл сжимать хеш-значения. Для ответа на этот вопрос было произведено предварительное исследование возможности сжатия (грубая оценка степени сжатия без учета времени) обычного списка хеш-значений, полученных разными алгоритмами. Под степенью сжатия подразумевается размер данных до сжатия, деленный на размер данных после сжатия.

Общие начальные данные:

- Размер данных: ~ 85 Гб;
- Количество файлов: ~ 62 600;
- Размер кусков: 4 Кб;

Алгоритмы хеширования:

1. MD5[4]
2. SHA-256[5]
3. Murmur[6]
4. Adler-32[7]

Алгоритмы компрессии[8][9]:

1. bzip2
2. DEFLATE
3. DEFLATE64
4. LZMA
5. PPMd

Таблица 1. Алгоритм хеширования – MD5. Размер хеш-значений до сжатия – 617 МБ (647 999 967 байт).

Метод сжатия	Размер после сжатия	Степень сжатия
bzip2	306 МБ (321 709 088 байт)	**49,65%**
DEFLATE	320 МБ (336 216 050 байт)	51,89%
DEFLATE64	316 МБ (332 074 713 байт)	51,25%
LZMA	315 МБ (330 878 400 байт)	51,06%
PPMd	310 МБ (325 753 173 байт)	50,27%

Таблица 2. Алгоритм хеширования – SHA-256. Размер хеш-значений до сжатия – 1,16 ГБ (1 254 291 295 байт).

Метод сжатия	Размер после сжатия	Степень сжатия
bzip2	602 МБ (631 597 675 байт)	**50,35%**
DEFLATE	644 МБ (675 293 074 байт)	53,84%
DEFLATE64	611 МБ (641 635 478 байт)	51,16%
LZMA	612 МБ (642 071 349 байт)	51,19%
PPMd	608 МБ (638 177 962 байт)	50,88%

Таблица 3. Алгоритм хеширования – Murmur. Размер хеш-значений до сжатия – 217 МБ (228 084 578 байт).

Метод сжатия	Размер после сжатия	Степень сжатия
bzip2	84,1 МБ (88 259 695 байт)	38,70%
DEFLATE	92,4 МБ (96 975 855 байт)	42,52%
DEFLATE64	92,1 МБ (96 580 039 байт)	42,34%
LZMA	83,9 МБ (88 030 687 байт)	38,60%
PPMd	83,3 МБ (87 392 776 байт)	**38,32%**

Таблица 4. Алгоритм хеширования – Adler-32. Размер хеш-значений до сжатия – 235 МБ (246 453 959 байт).

Метод сжатия	Размер после сжатия	Степень сжатия
bzip2	82,6 МБ (86 663 017 байт)	35,16%
DEFLATE	87,7 МБ (92 063 117 байт)	37,36%
DEFLATE64	87,7 МБ (92 038 539 байт)	37,35%
LZMA	84,0 МБ (88 153 004 байт)	35,77%
PPMd	82,5 МБ (86 585 177 байт)	**35,13%**

3. Идентификаторы блоков

Кроме хеш-значений, нам требуется хранить некоторые метаданные: ссылки на файлы, по которым посчитаны хеши, а также номер блока файла, по которому посчитано данное хеш-значение. Получается, что ко всем хеш-значениям одного файла, будет относиться одна и та же ссылка – разница будет лишь в номере блока.

Ссылки могут быть двух видов:

- Путь к файлу вида "C:\...", "/mnt/..." и т.д. в виде строки;
- Уникальный идентификатор файла на партиции (например, номер первого NTFS extent (Microsoft Windows)[10] или inode (UNIX-системы)[11]);

Рассмотрим ссылки второго вида, т.к. они являются шестнадцатеричными числами, которые и до, и после сжатия имеют меньший объем по сравнению с ссылками первого вида. Проверим, насколько они поддаются сжатию.

Таблица 5. Степень сжатия ссылок – уникальных идентификаторов файла на партиции.

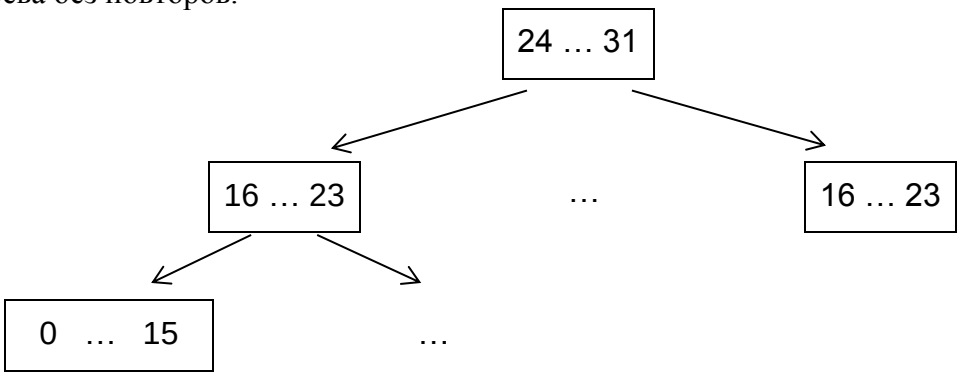
Метод сжатия	Степень сжатия
bzip2	32,16%
DEFLATE	34,25%
DEFLATE64	34,25%
LZMA	33,12%
PPMd	**31,89%**

4. Представление хеш-значений

Перед тем, как сжимать хеш-значения, можно представить их в более удобном и компактном для нас виде. Можно разделить каждое хеш-значение на несколько частей фиксированного размера, а затем представлять список хеш-значений в виде дерева без повторов. То есть при совпадении одной части у нескольких хешей, она записывается лишь один раз, а остальные части этих хешей станут ее потомками.

Допустим, мы имеем 128-битную хеш-функцию (рисунок 2). Соответственно, каждое хеш-значение состоит из 128 бит, что равно 32 байтам. Предлагается разделить каждый хеш на три части (количество уровней дерева может варьироваться, как и размер частей): 16 байт, 8 байт и 8 байт. Далее у каждого узла сохраняются лишь уникальные потомки.

Рисунок 2. Представление списка хеш-значений 128-битной хеш- функции в виде дерева без повторов.



К примеру, если взять 128-битную хеш-функцию MD5 и представить ее в виде такого дерева (2 уровня: 8 байт и 24 байта), то для 6 миллионов хешей экономия места составит 0,4% лишь на том, что мы по-другому представили хеш-таблицу. Стоит отметить, что этот процент будет расти в зависимости от количества хеш-значений – при увеличении количества хеш-значений растет количество повторов их частей.

5. Коллизия хеш-функций

В хеш-функциях, которые возвращают хеш-значение постоянной длины, коллизий невозможно полностью избежать, поскольку хотя бы для одного значения хеш-функции соответствующее ему множество входных данных будет бесконечно, и любые два набора данных из этого множества образуют коллизию.

Оценим вероятность возникновения коллизий при дедупликации. Для удобства будем считать, что коллизии не вызываются искусственно, а значит, можно считать, что значения хеш-функции распределены равномерно. Пусть N – количество блоков в хранилище (напомним, что при записи файлы разбиваются блоки, по которым считаются хеш-значения), H = 2^h – количество значений хеш-функции (h – размер хеш-значения в битах), 2^{-h} – вероятность коллизии для двух блоков, P(N, H) – вероятность отсутствия коллизий. Тогда

$$P(N,H) = 1\left(1-\frac{1}{H}\right)\left(1-\frac{2}{H}\right)\ldots\left(1-\frac{N-1}{H}\right) = \frac{H!}{H^N(H-N)!}$$

Пусть N << H. При малых x можно считать, что 1 + x ≈ e^x. Тогда получаем

$$P(N,H) \approx e^{-\frac{1}{H}}\ e^{-\frac{2}{H}}\ldots e^{-\frac{N}{H}} = e^{-\frac{N(N+1)}{2H}}$$

$$P(N,H) \approx 1-\frac{N^2}{2H}$$

Таким образом, вероятность коллизии приближенно квадратично зависит от размера хранилища и примерно экспоненциально от размера хеша[12].

Рассмотрим конкретный пример:

- Размер хранилища – 128 Тб;
- Размер блока – 4 Кб;
- Размер хеша – 256 бит;
- N = 2^{35};
- H = 2^{256};

$$P(N,H) \approx 1-\frac{2^{70}}{2^{257}} \approx 1 - 5{,}1 * 10^{-56}$$

6. Обработка коллизий

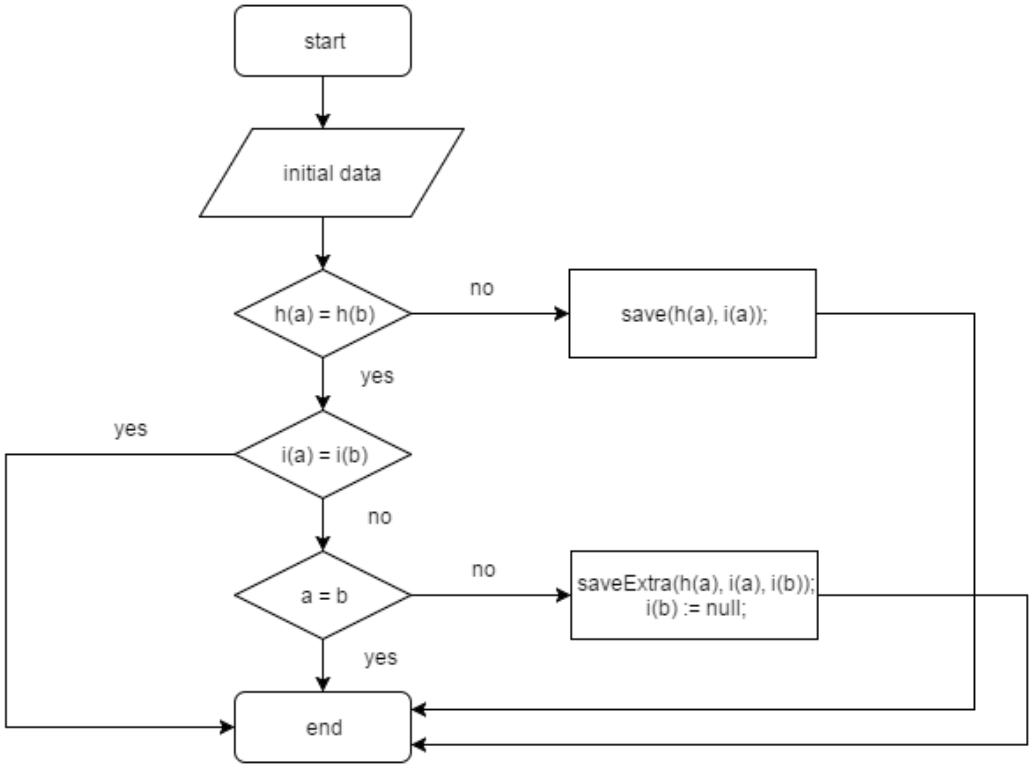
Размеры хранилищ данных могут сильно варьироваться. Исходя из этого, для экономии памяти на хеш-значениях при возможности стоит либо выбирать хеш-функции разных размеров для разных хранилищ, либо использовать лишь часть байтов от имеющихся хеш-значений. Для этого достаточно знать примерный размер хранилища данных, а также вероятность коллизии, которая нас устроила бы. Затем по приведенной в предыдущем параграфе формуле, нужно высчитать подходящий размер хеша.

Исходя из того, что мы сами выбираем удовлетворяющую нас вероятность коллизии (мы хотим сделать эту вероятность достаточно маленькой, но не слишком, чтобы не тратить чересчур много памяти на хранение хешей), а также при условии, что коллизии не вызываются искусственно, можно вместо сложной обработки коллизий использовать следующий метод:

При возникновении коллизии вместо идентификатора блока будем хранить нулевое значение. А для всех таких хеш-значений, у которых на месте идентификатора нулевое значение, дополнительно создается таблица с двумя полями: хеш-значение и идентификатор. Но в данной таблице хеш-значения уже необязательно должны быть уникальны. То есть в случае совпадения хеш-значений двух разных блоков в основной хеш-таблице будет храниться это хеш-значение и нулевой идентификатор, а в дополнительной хеш-значение и два разных идентификатора, которые указывают место хранения этих двух блоков.

Приведем блок-схему (рисунок 3) данного алгоритма, где a и b – блоки данных; h(a) – новое хеш-значение, посчитанное по блоку a; h(b) – старое хеш-значение, с которым сравниваем, посчитанное по блоку b; i(a) – идентификатор блока a; i(b) – идентификатор блока b).

Рисунок 3. Блок-схема алгоритма обработки коллизий при дедупликации.



7. Результаты эксперимента с данными

Процесс дедупликации был реализован на языках программирования Java и C. Вычислительная машина имела следующие характеристики:

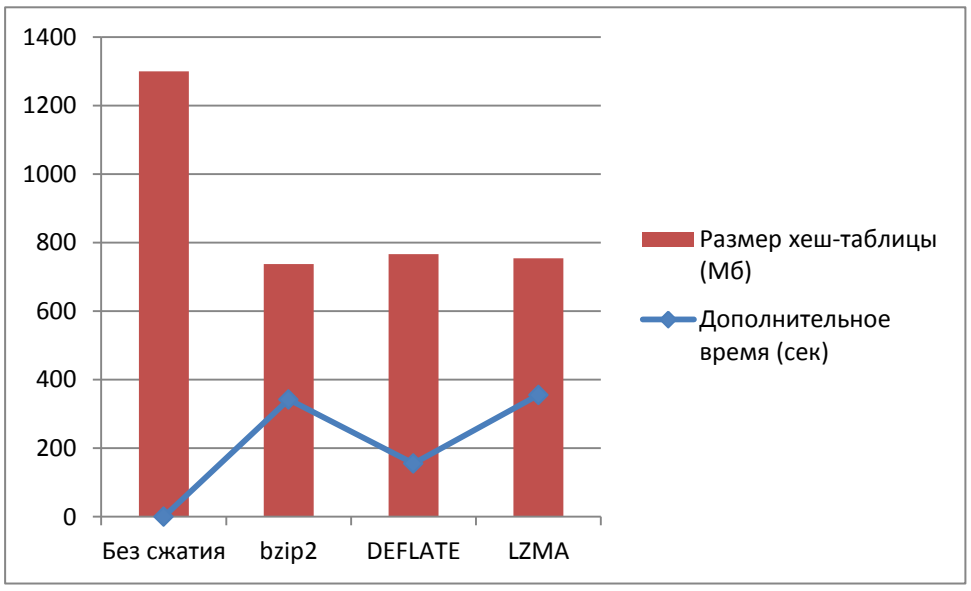
- Процессор: Intel(R) Core(TM) i3 2.3GHz;
- Оперативная память: 8GB;
- Операционная система: Windows 7 Home Basic;

Размер данных для эксперимента ~ 200 Гб. Далее при упоминании хеш-значений без сжатия имеется в виду обычная хеш-таблица. А после сжатия хеш-таблица уже имеет вид, предложенный в параграфе 4.

Алгоритм хеширования – MD5; Размер хеш-значений без сжатия ~ 1300 Мб; Размер идентификаторов блоков ~ 350 Мб;

a. Алгоритм сжатия – bzip2; Размер хеш-таблицы после сжатия ~ 737 Мб; Дополнительное время, которое потребовалось на сжатие и разжатие блоков при дедупликации ~ 342 сек;
b. Алгоритм сжатия – DEFLATE; Размер хеш-таблицы после сжатия ~ 766 Мб; Дополнительное время, которое потребовалось на сжатие и разжатие блоков при дедупликации ~ 155 сек;
c. Алгоритм сжатия – LZMA; Размер хеш-таблицы после сжатия ~ 754 Мб; Дополнительное время, которое потребовалось на сжатие и разжатие блоков при дедупликации ~ 355 сек;

Рисунок 3. Гистограмма зависимости дополнительного времени процесса дедупликации и размера хеш-таблицы от алгоритмов компрессии.



8. Заключение

В данной работе была рассмотрена задача сжатия хеш-значений в процессе дедупликации. В ходе решения данной задачи были произведены следующие работы:

- Изучение литературы по алгоритмам хеширования и компрессии;
- Разработка структуры данных для дедупликации, которая позволит хранить хеш-таблицу к более компактному виду;
- Разработка программного кода и проведение экспериментов на его основе;
- Сравнение результатов для разных алгоритмов сжатия;

Результаты эксперимента (параграф 7) показывают, что уменьшение затрат на хранение таких метаданных, как хеш-значения и идентификаторов блоков, действительно возможно, и умеренно увеличивает время процесса дедупликации.

В качестве продолжения работы можно предложить реализацию, в которой при переполнении имеющейся in-memory базы данных, часть хеш-значений отправляется во вторичную память.

Список литературы

[1] P. Shilane, M. Huang, G. Wallace, and W. Hsu. WAN Optimized Replication of Backup Datasets Using Stream-Informed Delta Compression. In Backup Recovery Systems Division EMC Corporation

[2] Data deduplication. http://www.emc.com/corporate/glossary/data-deduplication.htm

[3] Ахо, Хопкрофт и Ульман. Структуры данных и алгоритмы. 2003. С. 116-121

[4] R. Rivest. The MD5 Message-Digest Algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc. 1992

[5] Penard W, Werkhoven T. On the secure hash algorithm family. 2008

[6] MurmurHash. http://murmurhash.googlepages.com/

[7] Theresa Maxino. Revisiting Fletcher and Adler Checksums Carnegie. Mellon University Student Forum. 2006

[8] Алгоритмы сжатия данных без потерь. http://habrahabr.ru/post/231177/

[9] Алгоритмы сжатия данных без потерь. Часть 2. http://habrahabr.ru/post/235553/

[10] Understanding Pages and Extents. https://msdn.microsoft.com/en-us/library/ms190969.aspx

[11] Richard E. Smith. Elementary Information Security. 2013. С. 203-205

[12] А. Кладов. Презентация “Надежность хеша для однозначной идентификации данных при дедупликации”. 2011