УДК 004.632.4

**Дедупликация данных в распределенных объектных хранилищах**

***А. В. Хацкевича***

Московский физико-технический институт (государственный университет)

# На чём основывалось исследование

Исследования, проведенные в рамках данного доклада базируются на работе, освещенной в статье [ExtremeBinning]. В статье рассказывается о так называемой технологии связывания данных, которая предполагает использование подобия файлов для увеличения производительности дедупликации.

# Связывание данных

Связывание данных представляет из себя процесс нахождения похожих файлов и дальнейшее их обьединение, с целью хранить их рядом и не хранить их одинаковые части. Технология изначально рассчитана на огромные количества данных, и является способом борьбы с ростом накладных расходов на дедупликацию бэкапов с ростом объема данных.

Основная проблема дедупликации данных, по мнению авторов статьи, заключается в том, что при попытке выделять одинаковые куски файлов, возникает состояние, в котором один файл является разбросанным по множеству машин, и его сборка представляет из себя трудоемкий и ресурсозатратный процесс, в котором абсолютно не используется последовательный доступ, для которого оптимизировано большое количество оборудования используемого сегодня. Это приводит к деградации производительности до неприемлемой.

*Иллюстрация 1: Применение подхода extreme binning*

Блоки различных цветов изображают куски от различных файлов.
а) Слева условно изображено распределение кусков различных файлов при дедупликации без оптимизации.
б) Справа условно изображено распределение кусков различных файлов при дедупликации с применением подхода extreme binning.

Решение проблемы, заключается в том, что при добавлении нового файла бэкапа, ищется файл, наиболее похожий на новый, и происходит их слияние, причём таким образом, чтобы куски этих файлов оказались близко друг к другу. Последующие, похожие файлы — кладутся туда же. Это приводит к тому, что дедупликация происходит не между всеми файлами, а только между похожими. Это в свою очередь приводит к уменьшению уровня дедупликации. Однако исследование проведенное в статье показало, что в случае бэкапов, речь идет лишь о величинах порядка 20% от максимального уровня дедупликации.

Одна из применяемых оптимизаций — нахождение вероятных границ одинаковых кусков. Есть два наиболее известных алгоритма, решающих эту задачу: скользящее окно и tttd.

В статье был предложен следующий способ отыскания подобных файлов:

Разбитие файла на куски для каждого файла является операцией независящей от других файлов. Поэтому новый файл бьется на куски и рассчитывается криптографический хэш от каждого

куска. Далее, каким либо образом выделяется так называемый представительный (representative) кусок по особому правилу, максимизирующему вероятность совпадения представительных хэшей похожих файлов. В качестве наиболее простого правила было выбрано нахождение минимального из всех хэшей кусков файла.

## *Предложенная архитектура хранения данных*

В работе было рассмотрено несколько способов хранения данных ( кусков). Остановимся подробнее на каждом из них:

## Сохранение кусков в обычном объектном хранилище (T1)

Принцип работы таков: происходят обычные операции дедупликции, далее каждый кусок сохраняется в объектном хранилище как простой объект.

Стоит отметить:

- Никак не используется локальность данных[1].
- Излишнее количество метаданных[2].
- Удаление кусков происходит очевидным образом[3].

## Сохранение кусков в распределённой файловой системе как файлов (T2)

В этой модели на распределенной файловой системе для каждого сборного объекта создается папка, в которой хранятся куски принадлежащие всем объектам, относящимся к данному сборному объекту.

Остается не решенным вопрос о том, что типичный размер файла сравним с размером блока, что создает неявные расходы на память (не для всех файловых систем).

Файловая система не лучшим образом адаптирована для работы с миллионами файлов.

Из положительных моментов - такой подход, хоть и не в полной мере, но использует локальность данных. Это значит, что запрос на извлечение объекта будет работать внутри одной папки, которая в свою очередь с высокой вероятностью будет закэширована на ответственной за этот файл машине.

## Сохранение кусков в распределённой файловой системе в файлы, содержащие куски примерно равных размеров (T3)

Такой подход к хранению структур малого размера широко используем при работе с небольшими файлами, потому что он позволяет использовать локальность, быстро выделять место для новых структур, следить за свободным местом и удалять структуры без дополнительных потерь производительности и места.

В рамках объектного хранилища с дедупликацией предлагается использовать следующую структуру:

Для каждого сборного объекта в распределенной файловой системе создается папка, в которой хранятся файлы с кусками одинакового ( близкого) размера (ФСКОР). ФСКОР содержит в себе куски,

1 Подход extreme binning говорит о том, что мы должны хранить данные похожих файлов рядом. В случае с объектным хранилищем мы не можем повлиять на то, где будет сохранен кусок.
2 Нижележащее объектное хранилище хранит полный набор метаданных для куска как для полноценного объекта, к которому получают доступ не только из внутренних серверов.
3 Удаление куска из нижележащего объектного хранилища.

размер которых варьируется в небольших пределах. Поскольку размер куска ограничен сверху и с низу, ФСКОР-ов различного размера потребуется разумное количество, зависящее от того, в каких пределах варьируются размеры кусков внутри них. Расположение кусков во ФСКОР-ах выровнено. Назовем дельтой разницу между самым маленьким и самым большим кусками, которые можно сохранить в один ФСКОР. Дельта в общем случае может зависеть от размера куска. Размер дельты влияет на эффективность хранения данных. Далее для базовых тестов использовалась дельта фиксированного размера.

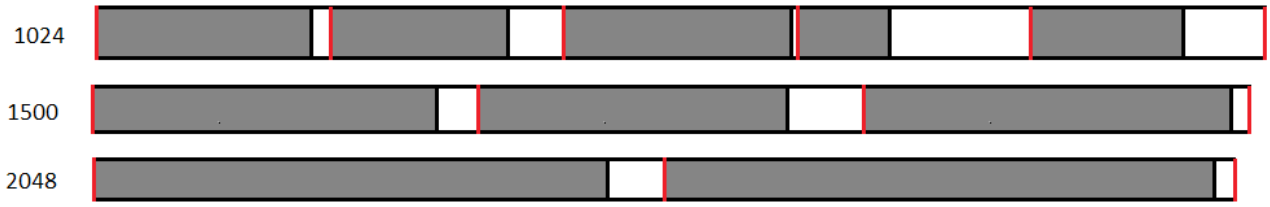



*Иллюстрация 2: Расположение кусков во ФСКОР-ах*
Различные линии на рисунке изображают различные ФСКОР-ы.
а) серым цветом обозначены данные
б) белым цветом обозначены неиспользуемые участки
в) границы выравнивания кусков обозначены красным

Выравнивание кусков необходимо для эффективного удаления объектов (их кусков). Это позволяет без труда дефрагментировать память и заполнять пробелы новыми кусками.

Стоит отметить, что хранение файлов в таком виде требует дополнительных метаданных.

Преимущества и недостатки:

- Почти максимальное использование локальности[4]
- Рациональное использование дискового пространства[5]
- Необходимость хранить и поддерживать информацию о том, где расположен конкретный кусок[6]

Этот вариант был выбран как основной при создании прототипа и проведении тестов.

# Тестирование прототипа

Стоит отметить что все тесты имеют качественный характер.

Тесты проводились на ноутбуке с операционной системой OS X, с 16 Гб оперативной памяти, SSD и процессором core i7 2,8 ГГц.

## *1. Скорость отдельных стадий дедупликации*

Для более наглядного представления производительности и профилирования различных подходов, было решено измерять различные стадии дедупликации по отдельности.

Вот таблица [1] полученных результатов (тест для файла 1.02 Гб):

4 Создается довольно большое количество файлов (количество зависит от дельты), и куски принадлежащие одному файлу не обязательно лежат последовательно.
5 Рациональность зависит от дельты. Увеличение дельты приводит к увеличению локальности, но к менее рациональному использованию дискового пространства.
6 Это может представлять из себя проблему при поддержке одновременного доступа к этим файлам с нескольких серверов, а так же при перераспределении шардингов на живую.

| Скорость (мб/с) | Кусок в другой ос | Кусок как файл | Кусок в файле кусков одного размера |
|---|---|---|---|
| Бегущего окна | 20.128 (C 242.27) | | |
| md5 | 274.8 | | |
| Сохранения метаданных | 18.035 | | |
| Извлечения метаданных | 1164.6 | | |
| Сохранение уникального объекта | ~ 0.1 | 1.735 (47.58) | 336.91 |
| Сохранение эдентичного объекта | ~ 0.1 | 1458.3 | 3767.3 |
| Извлечения данных | ~ 0.1 | 167.9 | 533.3 |

*1. Таблица: 1*

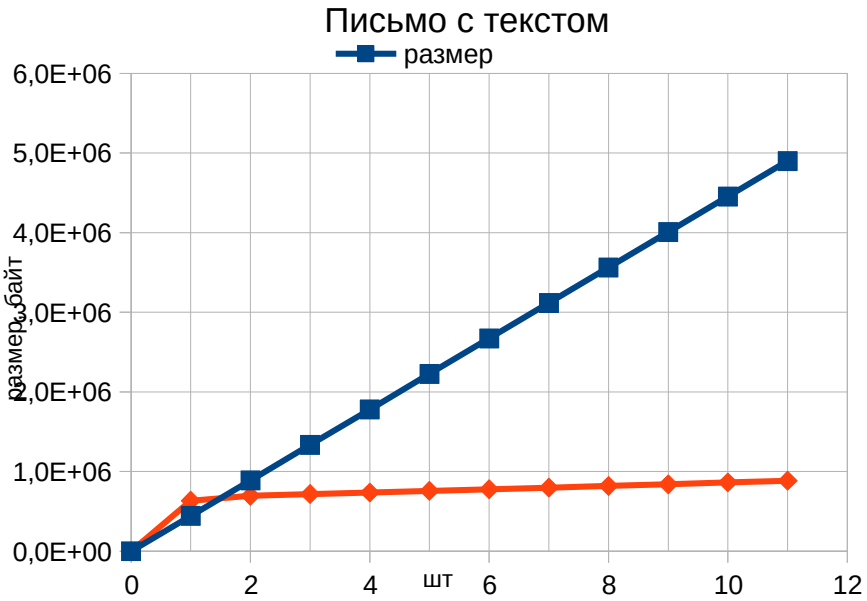
Интересно отметить, что бутылочным горлышком оказалось не только добавление записей в базу данных, но и работа алгоритма «бегущее окно». Стоит отметить, что реализация этого алгоритма в прототипе, написанном на C имеет скорость 240 мб/с. Вероятно причиной тому послужила классовая структура кода написанного на java и большое количество условных операций. В любом случае, существует возможность реализовать эту процедуру на C при помощи JNI.

Метод «кусок в файле кусков одинакового размера» оказался значительно производительнее и не вызывал замедления системы.

## 2. Дедупликация писем электронной почты

Этот тест так же носит качественный характер.

В ходе теста было решено попробовать продедуплицировать письма электронной почты, т. к. такой вид данных является предположительно важной областью применения объектного хранилища с дедупликацией.




*1. График*

Для проведения теста, письмо было переотправлено множество раз между двумя почтовыми ящиками. Тестировались письма со вложениями и без. Текстом в обоих случаях выступала статья с

википедии «ро-алгоритм Полларда», размером 40 кб.

Письма были сохранены почтовым клиентом в формате исходных данных.

Как видно из графика [1], первое письмо занимает больше места в случае с дедупликацией. Прирост занимаемого места составляет около 30 кб в обоих случаях. Далее, дедупликация начинает переиспользовать повторяющиеся куски и с каждым новым письмом, требуется лишь незначительное количество места.

### *3. Потеря дискового пространства связанная с соразмерностью кусков и блоков файловой системы*

Был замерен эффективный размер занятого места для двух подходов сохранения кусков:

| **Скорость (мб/с)** | Кусок как файл | Кусок в файле кусков одного размера |
|---|---|---|
| Занято места | 1,02 | 1,06 |
| Занято места эффективного | 1,31 | 1,07 |

*2. Таблица: 2*

Таким образом большое количество файлов [2] приводит не только к значительной потере производительности, но так же и к неявной потере дискового пространства.

## Выводы:

Тесты 1-3 показали, что из трех подходов сохранения данных, подход, в котором использовались ФСКОР-ы является лидирующим по всем ключевым показателям.

Дедупликация становится выгодной с точки зрения занимаемого места только при наличии нескольких похожих файлов. Дедупликация так же требует дополнительных расходов вычислительных мощностей и интенсивной работы диска. Поэтому, вероятно, имеет смысл не всегда дедуплицировать объекты. Мысль для дальнейших исследований такова: вычислять представительных хэш объекта при сохранении, и дедуплицировать его только при поступлении нового объекта с таким же хэшем.

## Литература

1. AmazonS3: amazon, "Amazon Web Services Offers European Storage for Amazon S3" (Press release), 2007, phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol- newsArticle&ID=830816

2. ExtremeBinning: Deepavali Bhagwat, Kave Eshghi, Darrell D. E. Long, Mark Lillibridge, Extreme Binning, 2009,Лондон, ISBN 1526-7539

3. GluterFS: gluter.org, GluterFS, , gluster.org/community/documentation/index.php/Main_Page

4. Finite Mathematics: John G. Kemeny, J. Laurie Snell, Gerald Thompson, Introduction to Finite Mathematics, издание 3, 1974, ISBN 978-0134838342