Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Московский физико-технический институт (государственный университет)»

Факультет радиотехники и кибернетики

Кафедра « Теоретической и прикладной информатики»

Хацкевич Алексей Владимирович

Дедупликация данных в распределённых объектных хранилищах

Выпускная квалификационная работа бакалавра

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:

Студент 211 группы _____ **Хацкевич Алексей Владимирович**

Научный руководитель _____ **Иваничкина Людмила Владимировна**

г. Москва

2016

Оглавление

Введение

Об авторе

Данный диплом написан Хацкевичем Алексеем Владимировичем, учеником 4 курса Московского Физико-Технического Института (государственного университета), на кафедре теоретической и прикладной информатики (закрепленной за компанией parallels).

Кратко о предметной области

В последнее время разработка новых продуктов в сфере IT очень упростилась. Сейчас сделать качественный продукт в состоянии даже небольшая компания. От части это происходит благодаря тому, что для выполнения определенных задач уже разработана определенная база технический решений высокого качества. Используя такие решения, небольшие компании экономят время на разработке и поддержке продукта. За частую при использовании таких решений, не обязательно даже представлять как оно устроено внутри. Один из таких продуктов - так называемое объектное хранилище.

Объектное хранилище представляет из себя абстракцию хранилища данных, позволяющую не заботится о внутреннем устройстве хранилища, одновременно обеспечивая высокую производительность, надежность, отказоустойчивость. В отличии от файловых систем, такая архитектура не является иерархической, она лишь хранит так называемые "объекты" ассоциируя их с "ключами". Это означает что у пользователя есть возможность сохранить данные по определенному ключу, а затем достать их по тому же ключу.

В силу сложности реализации надежных объектных хранилищ, зачастую ограничиваются рядом простых операций, доступных в рамках системы.

Основные из них[AmazonS3]:

- добавление объекта
- удаление объекта
- получение объекта

Помимо этого, предоставляется REST API и ACL. Все это дает возможность хранить таким образом неупорядоченные данные с возможностью их запроса пользователем минуя основные сервера сервиса.

Сегодня существует большое количество подобных продуктов, и многие их них унаследовали протокол взаимодействия схожий с таковым у amazon S3

storage. Хранилища с аналогичным api называются «amazon S3 compatible stprage». Однако api описывает лишь то, каким образом отправлять данные на сервер, получать их от туда, а так же взаимодействие с правами доступа. В api не указывается, каким образом хранятся данные непосредственно на сервере. Это дает определенную свободу при реализации своего объектного хранилища. В частности, это дает возможность использовать алгоритмы сжатия и дедупликации для непосредственного хранения информации.

Цель диплома

Целью данного диплома является исследование возможностей создания распределенного объектного хранилища со встроенной дедупликацией на основе существующего объектного хранилища и распределенных файловых систем, используя подход extreme binning. Проведение качественного сравнения различных способов хранения данных. Выработка общей принципиальной схемы устройства хранилища. Исследование ограничений, возникающих в объектном хранилище при введении дудупликации.

На чём основывалось исследование

Исследования, проведенные в рамках данного диплома базируются на работе, освещенной в статье extreme binning[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, как уже было сказано, 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.

Наряду с общей концепцией в статье предлагается алгоритм нахождения похожих файлов и упоминается алгоритм поиска похожих кусков в этих файлах.

Нахождение одинаковых кусков в файлах

Предположим, что мы нашли похожие файлы. Перед нами стоит задача отыскать похожие куски в этих файлах.

Одинаковые куски — это пары одинаковых массивов байт в двух файлах, которые могут располагаться на разных местах. Для них не задается строго длинна. Она может варьироваться в пределах максимального и минимального значений, при этом имея параметр наиболее вероятной длинны, определяющиеся спецификой аппаратной платформы и спецификой данных, для достижения максимальных показателей дедупликации.

Очевидно, что лобовое решение в виде простого сравнения всех возможных кусков одного файла со всеми возможными другого представляется ресурсозатратным и не может быть применено.

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

Задачей такого алгоритма является нахождение вероятных границ кусков таким образом, чтобы они были устойчивы к изменениям объекта. Это значит, что небольшое изменение объекта должно приводить к минимальному изменению кусков. В идеале, меняются только куски затронутые изменением, а границы близких к ним кусков остаются без изменений.

Скользящее окно

Создается так называемое окно — промежуток байт из файла, заданной длинны, который перемещается по файлу. Для каждого положения окна рассчитывается некий хэш и на его основе принимается решение о том, что это положение окна может быть началом или концом одинакового куска (для которого существует идентичный кусок в похожем файле).

Нам не обязательно сравнивать хэши одной границы с хэшами границ из другого файла, нам достаточно найти в обоих файлах границы с хэшами из определенного набора значений, и, вероятно они совпадут между собой. Эта оптимизация сильно увеличивает производительность, и поэтому очень важна, однако она приводит к тому, что мы не самым оптимальным образом находим похожие куски. Похожие куски могут дробиться, или не находиться вовсе. Но оказывается, что при объединении более чем двух файлов, не возможно

обойтись без подобных издержек[1].

Так же, предлагается в качестве хэш-а использовать какой-либо из циклических хэшей. В качестве такого был выбран полиномиальный хэш Рабина.

После этого файл бьется по вероятным границам похожих кусков на куски и для каждого куска находится криптографически стойкий хэш. По совпадениям этих хэшей находятся похожие куски внутри группы похожих файлов (см. приложение).

Нахождение подобных файлов

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.

Таким образом похожий файл может быть быстро найден. Далее производится обьединение с ним нового файла. Такой подход допускает оптимизации последовательного доступа, локальности, а так же позволяет легко распределить данные по различным серверам. Единственное требование которое стоит перед распределитем — класть файлы похожих кусков на один сервер. Это не является сложной задачей. Однако даже отступление от этого не будет критичным, а всего лишь снизит качество дедупликации или её скорость[2].

1 Если два файла были идеально разбиты на куски, после добавления третьего файла, разбиение может перестать быть идеальным. Чтобы границы кусков остались наиболие оптимальными, необходимо, чтобы произошло переразбиение первых двух файлов. Однако это дорогостоящая операция.

2 В случае, если место на ноде полностью занято одной группой похожих файлов, можно либо ограничить количество похожих файлов в группах, либо раскидывать файлы из одной группы по разным нодам. В первом случае это обернется потерей дедупликации, а во втором — потерей производительности.

Построение архитектуры

Потребности потребителя

Объектное хранилище (OS) всё чаще является платформой, на базе которой создаются системы и сервисы. Это превращает хранилище в своего рода фундамент, предъявляя к нему такие требования как надежность, отказоустойчивость, скорость. Всем этим требованиям должно удовлетворять OS, чтобы создать адекватную абстракцию для вышележащих уровней.

Это должно быть учтено при построении архитектуры распределенного объектного хранилища со встроенной дедупликацией данных. В своих рассуждениях сфокусируемся на оптимизации архитектуры на таких операциях как:

- добавление объекта
- чтение объекта
- удаление объекта

В общем случае мы не располагаем информацией, которую хранит покупатель объектного хранилища. Это создает сложности для оптимизации[3].

Технологии хранения данных

В качестве хранилища кусков, для построения архитектуры было рассмотрено несколько вариантов существующих распределенных решений.

Существующее объектное хранилище

Так как исследование происходило в рамках некоторого набора уже существующих продуктов компании parallels, было интересно рассмотреть возможность создания объектного хранилища с дедупликацией на базе уже существующего объектного хранилища. То есть хранить куски файлов одного объектного хранилища в другом.

Использование существующего объектного хранилища для построения нового звучит заманчиво, однако порождает ряд проблем.

Объектное хранилище обладает скудным набором допустимых операций. Для очевидных схем работы хранилища с дедупликацией необходима операция модификации данных, которую плохо поддерживают объектные хранилища.

Другой, более значимой проблемой этого подхода является то, что нельзя

3 Некоторые типы данных не могут быть дедуплицированы, в таком случае объектное хранилище не должно вносить большой потери производительности по сравнению с хранилищем без дедупликации. Некоторые типы данных, наоборот, хорошо поддаются дедупликации. В таком случае хранилище сталкивается с проблемой разрастания групп похожий файлов, о чем так же необходимо позаботиться.

просто так хранить куски файлов как отдельные объекты вышележащего хранилища. В противном случае это приведет к тому, что связывание данных не будет работать и нельзя будет построить систему с адекватными показателями скорости. Для борьбы с этой проблемой необходимо создавать сложную структуру объекта, которая бы позволяла уменьшить количество изменений объектов, а так же упростить их удаление, при сохранении консистентности вышележащего объектного хранилища.

Дедупликация в классическом проявлении предполагает работу с большим числом файлов, а значит любое обращение к верхнему объектному хранилищу будет выраждаться в на несколько порядков большее количество обращений к нижележащему объектному хранилищу.

Придумать структуру объекта нивелирующую эти проблемы не удалось.

Преимущества данного подхода:

- используются собственные технологии
- нижележащее объектное хранилище обеспечивает распределённость и надежность

Недостатки данного подхода:

- нижележащее объектное хранилище имеет скудный набор допустимых операций (нет возможности изменить объект затратой разумного количества ресурсов)
- нельзя просто хранить куски файлов как объекты, т. к. тогда не будет использоваться локальность данных.

Распределенная файловая система

Рассмотрим как базис для построения объектного хранилища распределенную файловую систему.

Желаемые свойства системы:

- автоматическая репликация
- внутренние механизмы обеспечения консистентности
- кэширование

Такими свойствами обладают современные распределенные файловые системы[GluterFS]. Это дает возможность переложить часть ответственности на них. Есть возможность пользоваться примитивами файловых систем (папки, файлы), а так же адаптировать архитектуру для связывания данных.

Свойство кэширования позволит в динамическом режиме перераспределять данные между серверами.

Хранение метаданных

Метаданные — дополнительная информация о файлах, не являющаяся их содержимым. В них входят: название, время добавления, время изменения, служебная информация связанная с хранилищем, и т. д..

То, каким образом хранить метаданные, не имеет принципиального значения. В рамках объектного хранилища с дедупликацией, хранилище метаданных должно лишь отвечать следующим требованиям:

- обеспечивать быстрый доступ на любом из частых запросов (поиск кусков связанных с конкретным объектом)
- требовать памяти на порядки меньше хранимых данных
- быть отказоустойчивым
- быть масштабируемым
- легко переносить шардинг

Это типичная зада для современных баз данных (хоть и не решается средствами самой бд), и на этом мы не будем заострять внимание. В качестве инструмента хранения метаданных в дипломной работе была выбрана база данных postgresql.

Распределитель запросов

Распределитель запросов — сервер, или набор серверов, который знает к какому серверу обращаться за какими данными. Он занимается распределением запросов (проксированием, перенаправлением) к нужным серверам. Самый простой принцип разбиения данных по различным серверам — разбиение пространства хэшей от названий объектов по группам и сохранение групп на закрепленные под ними сервера. Так же не стоит забывать о проблеме масштабирования и перераспределения данных (шардинг). Однако этот вопрос не освещён в дипломе.

Предложенная архитектура мета данных

Для работы с дедуплицированными данными была предложена архитектура, состоящая из следующих примитивов:

Мета объект — сущность, хранящая основную мета информацию об объекте, ссылку на сборный объект.

Сборный объект — сущность хранящую мета информацию, связанную с одним набором похожих объектов.

Мета кусок — сущность хранящая в себе мета информацию об одном из кусков, а так же ссылку на сборный объект к которому он принадлежит.

Связь объекта и мета куска — набор из таких сущностей отображает объект на мета куски. Говорит о том, какой мета кусок куда поставить, чтобы собрать объект воедино.

Подразумевается, что для сохранения непосредственно куска используется информация из мета куска (криптографический хэш, идентификатор, идентификатор сборного объекта). Эта информация необходима для обеспечения свойства локальности, и её должен учитывать интерфейс реализуемый хранилищем данных.

Иллюстрация 2: связь метаданных

Таким образом, все куски на самом деле принадлежат какому-то из сборных объектов. Сами объекты собираются по кускам воедино при помощи специального отображения объекта на сборный объект. Такой принцип упрощает добавление последующих новых объектов и их мета информации.

Для добавления нового объекта необходимо:

1. разбить новый объект на куски
2. найти представительный кусок
3. найти интересующий нас сборный объект/создать новый, пустой
4. найти новые куски
5. сохранить их
6. добавить новые куски в сборный объект
7. создать отображение нового объекта на сборный объект посредством связи

Для поддержания мета-информации в актуальном состоянии, ведется подсчет количества ссылок на куски и на сборные объекты.

Удаление объекта должно приводить к тому, что количество ссылок на некоторые куски и сборные объекты станет равным нулю. Такие сущности можно удалять сразу, или с помощью сборщика мусора.

Предложенная архитектура хранения данных

В дипломной 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.

Не очевидная проблема связанная с этим подходом заключается в том, что объектное хранилище наверняка не адаптировано для хранения объектов размера порядка размера куска и размера блока. Это может привести к неявным расходам места, что критично, поскольку мы боремся за него, говоря о дедупликации.

Так же, стоит отметить, что для пропускной способности всего порядка 100 мегабайт/с необходимо совершать порядка 100000 запросов в секунду к нижележащему объектному хранилищу, что неприемлемо[4].

Стоит отметить:

- Никак не используется локальность данных[5].
- Излишнее количество метаданных[6].
- Удаление кусков происходит очевидным образом[7].

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

4 Гипотеза: при доступе к объектному хранилищу к объекту размером 1 кб (порядок размера куска) , на поиск объекта и установление соединения будет затрачено гораздо больше времени и ресурсов, чем непосредственно на отправку этого объекта.

5 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.

6 Нижележащее объектное хранилище хранит полный набор метаданных для куска как для полноценного объекта, к которому получают доступ не только из внутренних серверов.

7 Удаление куска из нижележащего объектного хранилища.

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 (Т3)

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, 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.

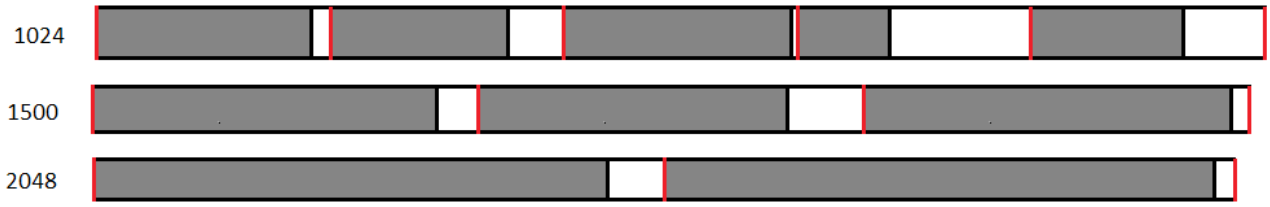


Иллюстрация 3: Расположение кусков во ФСКОР-ах
Различные линии на рисунке изображают различные ФСКОР-ы.
а) серым цветом обозначены данные
б) белым цветом обозначены неиспользуемые участки
в) границы выравнивания кусков обозначены красным

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. Было принято решение не хранить эти данные с другими метаданными, чтобы абстрагировать полностью инструмент хранения данных от остальных частей программы. В итоге смещение кусков во ФСКОР-ах хранилось в отдельных файлах в виде бинарной информации. Атомарные операции с такого рода представлением сложны в реализации. Создание

надежной архитектуры с репликацией потребует дополнительных усилий.

Преимущества и недостатки:

- Почти максимальное использование локальности[8]
- Рациональное использование дискового пространства[9]
- Необходимость хранить и поддерживать информацию о том, где расположен конкретный кусок[10]

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

Итог

Изначально предпочтительной архитектурой для объектного хранилища была архитектура, использующая другое объектное хранилище. К сожалению, из-за отсутствия в объектном хранилище операции изменения файла, а так же неприспособленности объектного хранилища быстро работать с огромным количеством маленьких файлов, не удалось придумать архитектуры, которая показывала бы достаточную производительность хотя бы в теории.

Методы работы с данными T2 и T3 обладают некоторыми преимуществами и недостатками, необходимо провести качественное тестирование.

Объектное хранилище со встроенной дедупликацией требует специализированного, заточенного под работу с маленькими файлами подхода к хранению данных. В контексте extreme binning, хранилище так же должно использовать локальность данных.

8 Создается довольно большое количество файлов (количество зависит от дельты), и куски принадлежащие одному файлу не обязательно лежат последовательно.

9 Рациональность зависит от дельты. Увеличение дельты приводит к увеличению локальности, но к менее рациональному использованию дискового пространства.

10 Это может представлять из себя проблему при поддержке одновременного доступа к этим файлам с нескольких серверов, а так же при перераспределении шардингов на живую.

Создание прототипа

Введение

В предыдущем разделе было объяснено какие решения и почему принимались. В этом разделе пояснениям будет уделено мало внимания. Цель этого раздела рассказать как работает прототип, подготовить читателя к разделу тестирования.

Всего было создано два прототипа, на C и на Java. Однако первый не имеет большого значения для диплома и обладает урезанным функционалом.

Выбор языка программирования

Так как изначально предполагалось использовать для хранения кусков объектное хранилище, а так же пользоваться базами данных, было решено использовать язык программирования имеющий развитое API.

Однако алгоритм дедуплицирования так же требует быстрых арифметических операций.

По вышеперечисленным причинам был выбран язык Java.

Создание базы для метаданных

В качестве хранилища метаданных была выбрана база postgresql, соотвественно была создана структура таблиц соответствующая описанной в разделе архитектура.

Для добавления объектов и кусков был создан набор внутренних транзакционных функций самой БД. К сожалению, пока не хватило времени создать механизм, добавляющий куски пачками, а не по одному. Это может стать бутылочным горлышком данного прототипа.

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 полученных результатов (тест для файла 1.02 Гб):

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

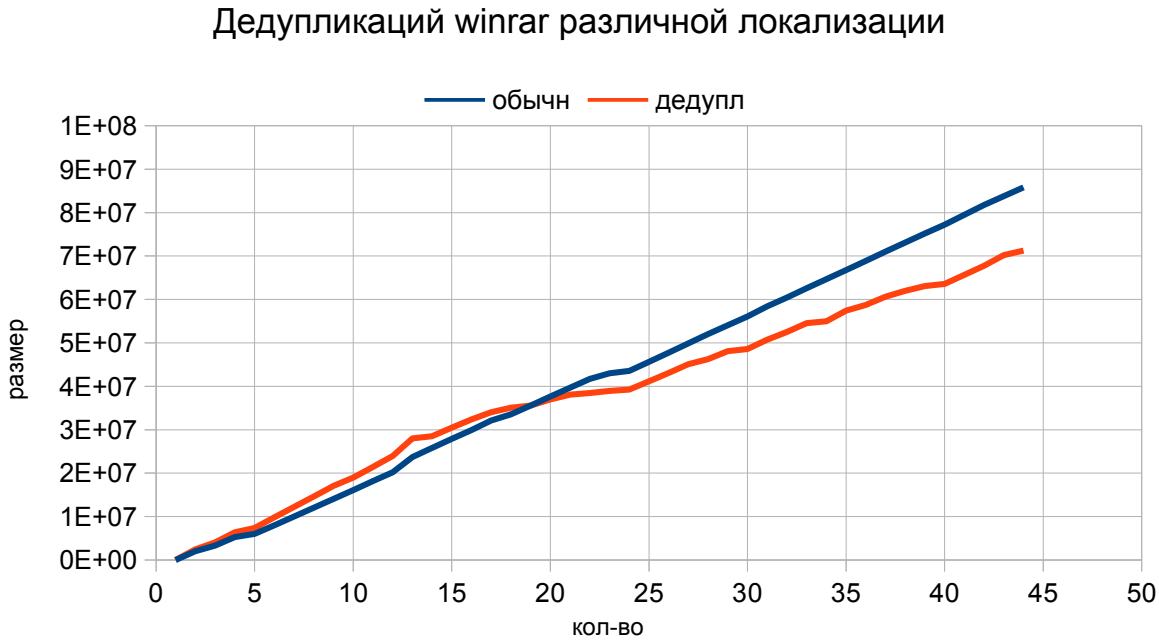
Интересно отметить, что бутылочным горлышком оказалось не только добавление записей в базу данных, но и работа алгоритма «бегущее окно». Стоит отметить, что реализация этого алгоритма в прототипе, написанном на C имеет скорость 240 мб/с. Вероятно причиной тому послужила классовая структура кода написанного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

В методе «кусок как файл» в начале сохранения объекта мы имеем скорость порядка 50 мб/с, однако она быстро падает. Средняя скорость записи файла размером 1,02 Гб оказалась равной 1,735 мб/с. Так же, хочется отметить, что после записи объекта наблюдалась огромная потеря производительности машины.

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

2. Дедупликация большого количества похожих файлов

В качестве большого количества одинаковых файлов были взяты бинарники winrar различной локализации. Был получен следующий график:



В процессе получения этого графика было отмечено, что крайне важную роль в дедупликации играют параметры скользящего окна. Варьируя их незначительно можно получить существенные изменения качества дедупликации. Это наталкивает на мысль, что метод скользящего окна не является самым подходящим для разбиения объекта на куски. В этом направлении требуются дополнительные исследования.

Так же хочется отметить, что дедупликация идентичных файлов, или файлов содержащих в себе одинаковый кусок несколько раз, происходит абсолютно без дополнительных расходов памяти.

На графике можно видеть, что дедупликация требует дополнительные издержки памяти, которые при недостаточном количестве похожих файлов приводят к увеличению занимаемого места по сравнению с исходным размером объектов.

С добавлением каждого следующего объекта, вероятность дедупликации увеличивается. В какой-то момент, дедупликация становится выгодной с точки зрения занимаемого места.

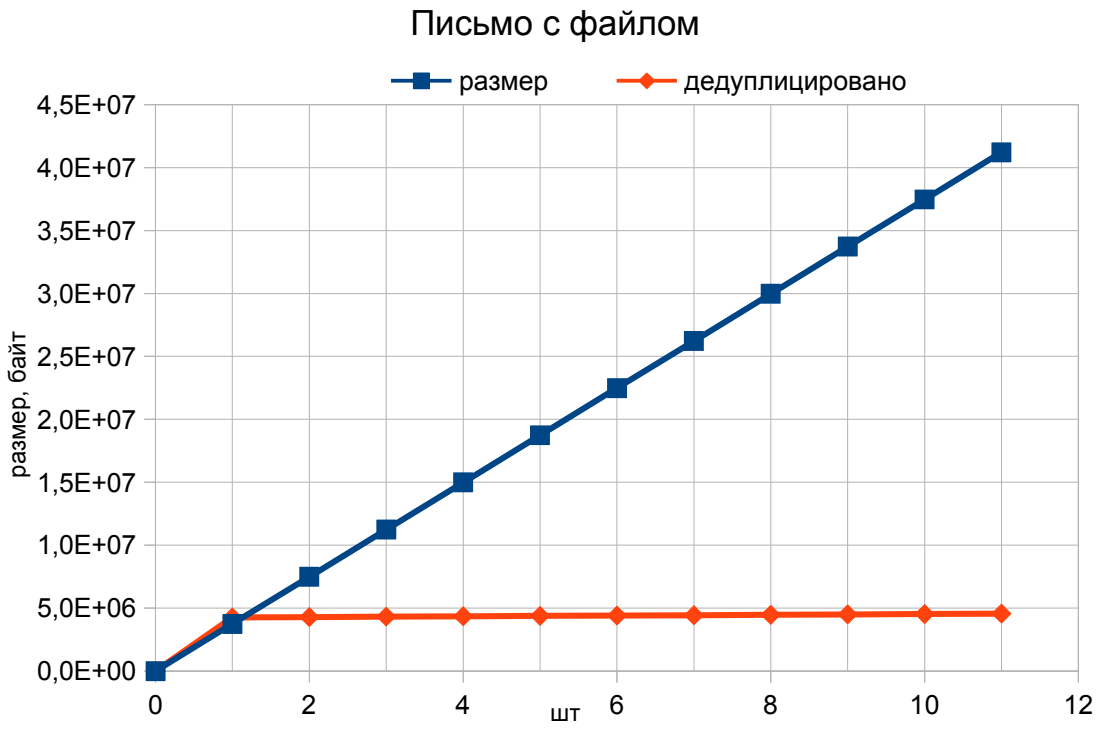
3. Дедупликация писем электронной почты

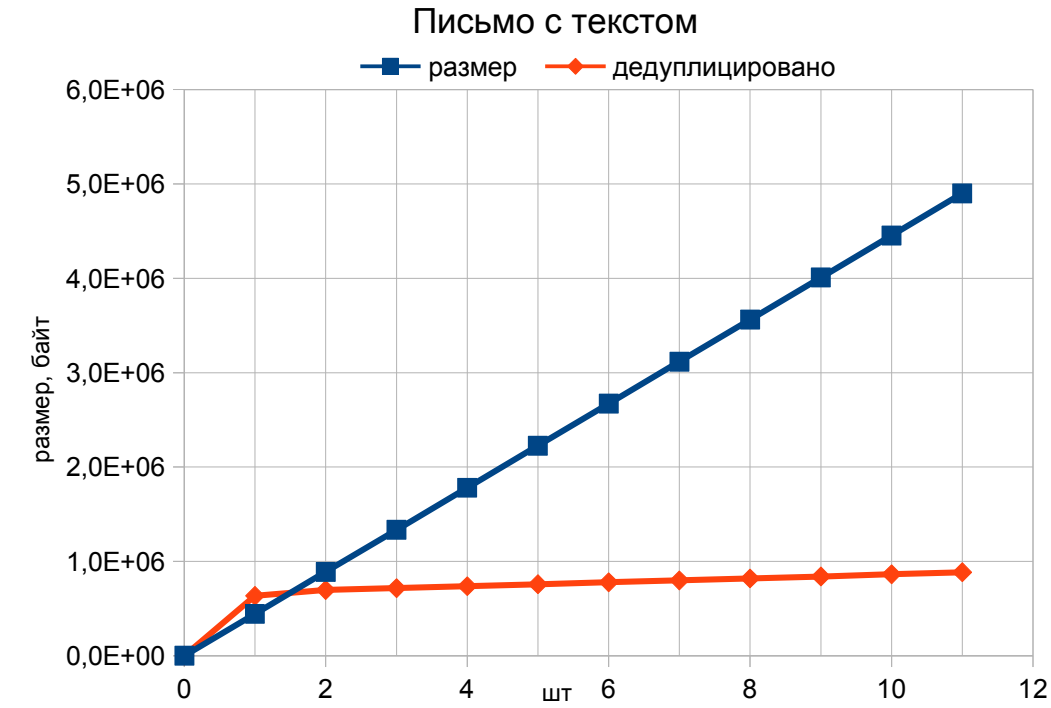
Этот тест так же носит качественный характер.

В ходе теста было решено попробовать продедуплицировать письма

электронной почты, т. к. такой вид данных является предположительно важной областью применения объектного хранилища с дедупликацией.

Вот результаты:





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 графиков, 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.

4. 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

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

Стоит отметить, что эта потеря зависит от минимального, среднего размера куска, от размера блока файловой системы. Эти параметры нельзя менять, не влияя на параметры дедупликации, поэтому хотелось бы избавиться от данной зависимости.

В случае с ФКОР мы имеем потери дискового пространства связанные с выравниванием кусков на дельту. На этот параметр проще влиять, подбирая необходимую дельту. Как уже было отмечено — дельта не обязана не зависеть от размера куска. Необходимо исследовать распределение кусков по размерам и найти оптимальное распределение дельт (см. приложение). Сильное уменьшение дельты может привести к неявным эффектам деградации производительности, вызванным работой программы с большим количеством открытых файлов одновременно. Эти эффекты требуют дальнейшего изучения.

Выводы:

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

Тест 2 показал, что 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.

Результаты проделанной работы

В ходе проделанной работы были получены следующие результаты:

- Изучены проблемы построения хранилища типа T1. Не придумана эффективная схема построения хранилища с дедупликацией на основе объектного хранилища.
- Создано и протестировано 3 подхода к хранению данных, проведено их качественное сравнение.
- Применен подход extreme binning к дедупликации в объектных хранилищах.
- Рассмотрены некоторые ограничения, возникающие в архитектуре объектного хранилища с введением дедупликации.

Дальнейшая работа

Выделяются такие направления дальнейших исследований, как:

- Выявление оптимальных минимального и среднего размеров куска
- Создание полноценного распределенного прототипа
- Решение проблемы шардинга
- Исследование влияния дельты на эффективность использования файловой системы

Основные понятия

Chunk, кусок — небольшой блок данных, размером около 2-40 кб, малая часть какого-то файла.

Объектное хранилище — хранилище данных, позволяющее сохранять и получать объекты по ключу.

Объект — один файл.

Распределенное хранилище — хранилище данных, работающее одновременно на нескольких логических компьютерах.

Окно — выделенный массив последовательных байт в файле, имеющий заданную длину, который перемещается по файлу.

Одинаковые куски — пары одинаковых массивов последовательных байт в двух файлах, которые могут располагаться на разных местах.

Файл с кусками одинакового размера (ФСКОР) — файл содержащий в себе куски примерно одинакового размера. Размещение каждого следующего куска выровнено на размер максимального куска для этого ФСКОР. Чтобы сохранить куски всех возможных размеров создается несколько ФСКОР.

Блок ФСКОР-а — промежуток байт размера куска максимальной допустимой длинны во ФСКОР-е. Всегда выровнен на свой размер.

Дельта — разница между самым большим и самым маленьким куском во ФСКОР-е. В общем случае зависит от размера блока ФСКОР-а.

Т1 — архитектура хранения данных объектов, в которой для хранения кусков используется объектное хранилище.

Т2 — архитектура хранения данных объектов, в которой для хранения кусков используются файлы и папки распределенной файловой системы.

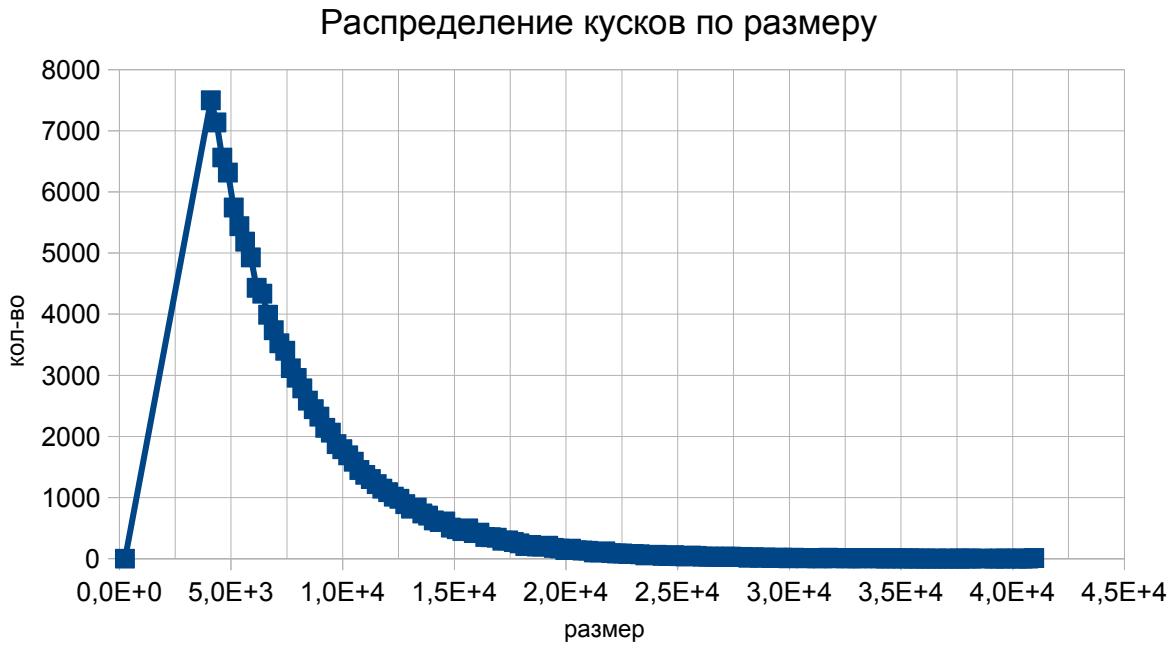
Т3 — архитектура хранения данных объектов, в которой для хранения кусков используется ФСКОР-ы и файлы и папки распределенной файловой системы, а так же файлы дополнительной мета информации.

Приложение

Распределение кусков по размерам

Распределение кусков по размерам — важная статистическая информация для дедупликации. Она не была исследована в основном дипломе, однако было проведено её измерение.

Такой график получается при сохранении единственного файла размером 1.02 Гб. Гранулярность группирования равна 256 б.



В дальнейшем предполагается использовать эту информацию для подбора функции дельты, для оптимизации использования эффективного места.

Коллизии хэшей кусков

Так выглядит ответ на обратную задачу поиска вероятности коллизии[Finite Mathematics]:

$$n(p,d) \approx \sqrt{(2 \cdot d \cdot \ln(\frac{1}{1-p}))}$$

- n — количество случайных чисел
- p — вероятность совпадения хотя бы двух
- d — мощность множества случайных чисел

Все расчёты имеют оценочный характер. Задача данного раздела — понять, стоит ли беспокоиться о такой вещи как коллизия хэшей.

Воспользуемся допущением о равномерности распределения значений

хэш функции.

За допустимую вероятность наличия в хранилище кусков с коллизией хэша возьмем, например 10^{-9} (меньше вероятности апокалипсиса).

1) Для начала рассмотрим обычную схему дедупликации, где все куски попадают в одно хранилище:

$$n \approx 8{,}24 \cdot 10^{14}$$

Это значение примерно соответствует объему хранилища в $4 \cdot 10^{6}$ петабайт.

2) Рассмотрим хранилище, работающее по принципу extreme binning:

Т.к. куски сгруппированы по сборным объектам, а их размер можно контролировать, то их количество в целом ограничено сверху, поэтому вероятностью коллизии на этом уровне можно пренебречь.

Точно найти вероятность коллизии представительного куска сложно, в силу того, что она сильно зависит от статистических данных по хранилищу, поэтому будем пользоваться грубыми оценками.

При равномерном распределении хэшей, выбор минимального хэша из всех хэшей кусков в схеме extreme binning уменьшает эффективное множество хэшей во столько раз, сколько кусков в объекте. Однако количеством случайных чисел уже будет выступать не количество кусков, а количество сборных объектов. За характерный размер объекта возьмем 100 мб.

$$n \approx 5{,}83 \cdot 10^{12}$$

Это значение примерно соответствует объему хранилища в $6 \cdot 10^{8}$ петабайт.

Источники

Литература

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

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

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

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

Изображения

Все изображения, графики и таблицы были нарисованы самим автором.