САНКТ-ПЕТРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(СПбГУ)

КАФЕДРА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

Курсовая работа на тему:

Исследование эффективности дедупликации с использованием цепочек преобразований при помощи разработанного инструментального средства

Выполнила:
студентка 345 группы
Екатерина Соса

Научный руководитель:
старший преподаватель кафедры системного программирования СПБГУ
Дмитрий Вадимович Луцив

2012

Оглавление

1 Введение в предметную область **3**

1.1 Понятие дедупликации данных . . . 3

1.1.1 Определение . . . 3

1.1.2 Важность темы . . . 3

1.1.3 Область применения . . . 3

1.2 Примеры решений по дедупликации . . . 4

1.3 Существующие задачи . . . 4

1.3.1 В связи с классификацией: потоковая и фоновая . . . 4

1.3.2 Дедупликация и архивация . . . 4

1.3.3 Способы определения дубликатов . . . 4

1.3.4 Оценка эффективности . . . 5

2 Постановка задачи **6**

2.1 Задача . . . 6

2.2 Задача как часть большой цели . . . 6

2.3 Актуальность . . . 6

3 Подход к решению **7**

3.1 Реализуемые алгоритмы . . . 7

3.1.1 LZ77 . . . 7

3.1.2 Интервальное кодирование . . . 7

3.1.3 Арифметическое кодирование . . . 7

3.2 Тестирование . . . 7

3.2.1 Эффективность в зависимости от параметров алгоритмов . . . 7

3.2.2 Эффективность в зависимости от типа данных . . . 8

3.3 Используемые технологии . . . 8

3.4 Ожидания . . . 8

3.4.1 Возможные проблемы . . . 8

3.4.2 Предполагаемые результаты . . . 8

4 Результаты **9**

4.1 Использование алгоритмов в зависимости от типа данных . . . 9

4.2 Оптимизация использования алгоритмов . . . 9

5 Приложения **10**
5.1 Описание алгоритмов . . . 10
5.1.1 LZ77 . . . 10
5.1.2 Интервальное кодирование . . . 10
5.1.3 Арифметическое кодирование . . . 10
5.2 Описание тестовых данных . . . 10
5.2.1 Тестовый набор 1 . . . 10
5.2.2 Тестовый набор 2 . . . 10
5.2.3 Тестовый набор 3 . . . 10
5.2.4 Тестовый набор 4 . . . 11
5.2.5 Тестовый набор 5 . . . 11
5.2.6 Тестовый набор 6 . . . 11
5.2.7 Тестовый набор 7 . . . 11
5.2.8 Тестовый набор 8 . . . 11
5.2.9 Тестовый набор 9 . . . 11

Литература **12**

Глава 1

Введение в предметную область

1.1 Понятие дедупликации данных

1.1.1 Определение

Дедупликация данных — это технология поиска и устранения дубликатов в хранилищах данных.
Процесс дедупликации можно разделить на три этапа: первый — подготовка данных, второй — работа с дубликатами, третий — сопровождение дедуплицированных данных. На каждом этапе возникают свои сложности, причём зависят они от многих параметров.

1.1.2 Важность темы

Технологию дедупликации уже давно активно применяют при резервном копировании и восстановлении. Кроме того, экономное хранение данных наряду с быстрым доступом к ним необходимо при виртуализации. Со временем дедупликацию захотелось рассматривать не только на фоне других целей и технологий, но и как отдельную задачу при размещении информации, её передаче, а также открытый вопрос: где ещё это может быть нужно? К тому же, сейчас количество информации даже на домашних машинах стало расти с такой скоростью, что пользователи просто не успевают самостоятельно отслеживать и устранять дублирующиеся данные, при этом потребность в свободном месте на дисках, а также в высокой скорости доступа не только имеет место, но и становится приоритетной. Обычно требуется удовлетворять в полной мере только одну из них, а это значит, что можно и нужно рассматривать частные случаи.

1.1.3 Область применения

Область применения технологии дедупликации на сегодня можно разделить на две: хранение данных, передача данных. Если смотреть на хранение, то это

- хранилища данных (архивы, статистика, бухгалтерия, медицина, резервное копирование, виртаулизация...)
- системы хранения пользовательского контента (закладки в браузере)
- рано или поздно, любой носитель.

Если смотреть на передачу, то это

- системы архивации

- системы репликации
- цифровое медиа-вещание
- рано или поздно, любой трафик

1.2 Примеры решений по дедупликации

1. Символьные ссылки Unix.
2. Opendedup (SDFS)
3. EMC Data Domain
4. ZFS

1.3 Существующие задачи

1.3.1 В связи с классификацией: потоковая и фоновая

Следует рассмотреть два понятия: «дедупликация на лету» и «дедупликация в фоновом режиме».
Под дедупликацией на лету подразумевается следующая ситуация: хранилище считается уже дедуплицированным, на него поступают данные, задача — разместить эти данные так, чтобы степень дедуплицированности осталась на уровне прежней. Есть различные подходы к решению этой задачи, о них стоит говорить отдельно, так же как и о том, в каких случаях тот или иной подход хорош. Под дедупликацией в фоновом режиме подразумевается другая ситуация: хранилище считается «заваленным» дубликатами (дедупликация либо вообще не проводилась, либо проводилась, но очень давно), задача — организовать хранение данных так, чтобы он было экономным, но не уменьшало доступности файлов. Тут, например, можно рассматривать случай, когда заранее известно, какие данные будут наиболее востребованы, в связи с этим использовать древовидные структуры для организации индексов, оставляя на верхних уровнях ту самую «нужную» информацию.

1.3.2 Дедупликация и архивация

Принято различать дедупликацию по размеру дедуплицируемых единиц: побайтовая (она же компрессия или архивация), блочная и файловая. Побайтовая дедупликация чаще всего используется для передачи файлов между пользователями по сети, при кодировании информации с целью её защиты, для хранения редко используемых данных на домашних компьютерах. Вопрос использования блочной и файловой дедупликации очень плохо изучен. Стоит понимать, что размер блока может быть и маленьким (меньше, чем средний размер файла в хранилище), и большим (больше, чем средний размер файла в хранилище). Также здесь возникает вопрос терминологии: что такое блок? Например, блок — это часть файла, размер блока — 2 МБ. В таком случае, каждый файл разбит на блоки по 2МБ и меньше (например, файл размером 5 МБ разобьётся на блоки 2 МБ, 2 МБ и 1 МБ). С другой стороны, существует и такое определение: блок — это блок файловой системы. Тогда внесём конкретику для дальнейшего общения. Будем рассматривать побайтовый, кусочный, файловый и блочный уровни дедупликации.

1.3.3 Способы определения дубликатов

Отдельный вопрос — это поиск дублей. Он стоит особняком, так как можно найти повторяющиеся фрагменты, но не устранять их. Есть точные способы (memcmp(bl1, bl2) == 0), они, зачастую медленные, но действи-

тельно точные: вероятность ошибки 0%, а есть, например, сравнение хэшей, что медленно только один раз, во время подсчёта, а дальше — сравнивается быстрее, но этот способ неточный, хотя допустимой ошибки может быть достаточно для конкретной задачи. Как ещё? Вопрос открытый для изучения и «живой», как отмечалось ранее.

1.3.4 Оценка эффективности

Оценку определённой реализации можно проводить по следующим критериям:

- масштабируемость (scalability)
- степень дедупликации (dedupe ratio)
- скорость дедупликации данных (data ingest rate)
- скорость доступа к данным (data access rate)
- структурная целостность (integrity)

Стоит пояснить пункт, характерный для этой области, но носящий, по большей части, рекламный характер: степень дедупликации — это величина, равная объёму данных в хранилище до дедупликации, делённому на объём данных в хранилище после. Вопрос оценки качества дедупликации тоже ещё не изучен, а главным критерием успешности алгоритма стоит назвать умение с его помощью решать поставленную задачу.

Глава 2

Постановка задачи

2.1 Задача

Оценить эффективность дедупликации посредством цепочек преобразований данных на тестовых наборах разных типов.

2.2 Задача как часть большой цели

Существует более общая задача - разработка инструментального средства для оценки эффективности дедупликации, при помощи которого и предполагается проводить тестирование.

2.3 Актуальность

На сегодня нет системы, позволяющей точно оценивать эффективность дедупликации на конкретных данных. Например, хорошо было бы поставить себе на компьютер какой-нибудь калькулятор, который покажет, как наиболее эффективно можно сжать те или иные каталоги. При этом испытание всех возможных преобразований занимает очень много времени, поэтому можно заранее провести исследования, чтобы, в зависимости от типа данных, подсказывать пользователю, какие алгоритмы применять и с какими параметрами.

Глава 3

Подход к решению

Для решения поставленной задачи разработано инстурментальное средство, для которого реализованы несколько алгоритмов сжатия, а также при помощи этого средства проведены исследования.

3.1 Реализуемые алгоритмы

3.1.1 LZ77

Этот алгоритим используется для сжатия в NTFS. (Подробное описание в Приложении)

3.1.2 Интервальное кодирование

Один из методов энтропийного кодирования, где можно менять единицу сжатия, то есть минимальный шифруемый элемент. (Подробное описание в приложении)

3.1.3 Арифметическое кодирование

Один из методов вероятностного сжатия, используя который, предполагается сжимать не только текстовые файлы, но и изображения. (Подробное описание в приложении)

3.2 Тестирование

В качестве тестов предполагается использовать каталоги с исходными кодами программ, в том числе совокупности каталогов разных версий одной и той же программы. Обычно, если брать каталог с достаточно большой вложенностью, данные теряют свою однородность, и среди файлов с исходными кодами встречаются картинки или другие нетекстовые файлы, поэтому хочется посмотреть, что будет, если каталог не полностью, а лишь почти полностью состоит из текстовых данных.

3.2.1 Эффективность в зависимости от параметров алгоритмов

Предполагается, что часть констант в реализациях алгоритмов целесообразно менять в зависимости от типа данных, их однородности, общего размера, необходимой скорости доступа и так далее, поэтому тестирование производится также и для определения оптимальных параметров работы алгоритмов.

3.2.2 Эффективность в зависимости от типа данных

Разрабатываемое инструментальное средство не умеет определять тип данных, которые поступают на вход алгоритму, но, применяя большое количество алгоритмов, можно посмотреть на статистику и выявить зависимость коэффициентов сжатия от типов данных. В дальнейшем эти знания можно применять для определения типа рассматриваемых данных.

3.3 Используемые технологии

Используется язык Python.

3.4 Ожидания

3.4.1 Возможные проблемы

Сложности возникают, потому что вычисления на Python производятся достаточно медленно, поэтому решено для тестов не брать каталоги больших размеров, но при этом количество тестов увеличить. Кроме того, сложно сравнивать алгоритмы, когда каждый из них зависит от какого-то своего параметра, поэтому решено сначала выбирать оптимальные параметры для каждого теста, а затем их сравнивать между собой, но вести параллельно другие ветви сравнений.

3.4.2 Предполагаемые результаты

Ожидается, что применение алгортимов сжатия со словарём для каталогов с исходными кодами программ будет эффективнее, чем дедуплицирование посредством хэширования. Кроме того, найдётся цепочка преобразований, которая будет эффективнее и на неоднородных данных, и на каталоге с изображениями.

Глава 4

Результаты

4.1 Использование алгоритмов в зависимости от типа данных

- Эффективнее интервальное кодирование, чем просто арифметическое на всех рассмотренных данных.
- LZO лучше, если нужна высокая скорость доступа, а LZ78 подходит для архивного хранения.
- Если тип данных заранее не известен, то завершением цепочки преобразований лучше брать интервальное кодирование.
- Алгоритм интервального кодирования на изображениях работает эффективнее в сочетании с вейвлет-преобразованием.
- В зависимости от эффективности сжатия рассматриваемых данных конкретными алгоритмами можно научиться определять тип этих данных.
- Использование LZ78 с общим для всего каталога словарём не даёт выигрыша на каталогах с исходными данными любых размеров.
- Для неоднородных данных использование LZ78 с общим для всего каталога словарём менее эффективно, чем использование LZ78 с отдельными словарями для всех файлов.
- Из всех LZ-алгоритмов самым неэффективными на тестах, где есть не только текст, оказался LZ78, там же, где кроме текста нет ничего, он наиболее эффективен, но с учётом оптимального параметра.

4.2 Оптимизация использования алгоритмов

- Рекомендованый размер словаря на неизвестных данных для LZ78 - 2**16
- Рекомендованый размер блока для блочной дедупликации на текстовых данных 512 байт или 256 байт
- Рекомендованый размер блока для блочной дедупликации на изображениях 2 килобайта
- Рекомендованый размер блока для неоднородных данных 512 байт

Глава 5

Приложения

5.1 Описание алгоритмов

5.1.1 LZ77

Алгоритм словарного сжатия, в котором в качестве словаря выступает множество подстрок из так называемого скользящего окна - блока символов некоторой длины (как правило, несколько килобайт), заканчивающегося в текущей позиции. Выходной поток разбивается на строки из словаря и кодируется в виде ссылок на них.

5.1.2 Интервальное кодирование

Модифицированный вариант арифметического кодирования, в котором вместо интервала вещественных чисел (0;1) используется интервал целых чисел от [0;N], и сообщение кодируется целым натуральным числом. Преимущество по сравнению с арифметическим кодированием заключается в простоте реализации.

5.1.3 Арифметическое кодирование

Разновидность энтропийного кодирования, при котором сообщение представляется в виде одного вещественного числа из диапазона (0;1). Обеспечивает близкую к теоритически воможной эффективность сжатия, позволяет, в отличие от алгоритма Хаффмана, использовать меньше одного бита на символ. Реализация алгоритма затрудненна из-за сложности представления произвольных вещественных чисел.

5.2 Описание тестовых данных

5.2.1 Тестовый набор 1

Три каталога с различными файлами исходных кодов на языке ассемблера. Размер - 19 403 789 байт.

5.2.2 Тестовый набор 2

Два каталога с исходными кодами двух версий программы GIMP. Размер - 276 022 016 байт.

5.2.3 Тестовый набор 3

Каталог с исходными кодами Open Office. Размер - 1 638 217 875 байт.

5.2.4 Тестовый набор 4

Два каталога с различными файлами исходных кодов на языке ассемблера. Размер - 10 399 969 байт.

5.2.5 Тестовый набор 5

Каталог с исходными кодами программы GIMP. Размер - 138 304 085 байт.

5.2.6 Тестовый набор 6

Четыре файла в формате pdf. Размер - 21 806 954 байт.

5.2.7 Тестовый набор 7

Каталог с изображениями. Размер - 1 335 211 байт.

5.2.8 Тестовый набор 8

Объединение данных из тестовых наборов 1 и 7. Размер - 20 739 000 байт.

5.2.9 Тестовый набор 9

Два каталога с версиями языка Python 2.6 и 2.7. Размер - 423 121 030 байт.

Литература

1. Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. Методы сжатия данных. Москва, ДИАЛОГ-МИФИ, 2003.
2. Carlos Alvarez. Руководство по установке и настройке дедупликации в системах NetApp FAS и V-Series. March 2009.
3. Christopher W. Fraser. An Instruction for Direct Interpretation of LZ77-compressed Programs. September 2002.
4. Chee-Yong Chan. An efficient bitmap encoding scheme for selection queries. 1999.
5. Ming Li and Yanong Zhu. Image Classification Via LZ78 Based String Kernel: A Comparative Study. 2006