САНКТ-ПЕТРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

Курсовая работа на тему:

Инструмент для оценки алгоритмов дедупликации

Выполнил: Алексей Кладов, студент 345 группа

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

2012

1. Введение

1.1. Определение дедупликации

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.

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. Кроме того, 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.

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. Рассматриваемые задачи

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.

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.

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 действительно 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. Решения по дедупликации

Символьные ссылки Unix
Opendedup (SDFS)
EMC Data Domain
ZFS

1.4. Оценка дедупликации

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)

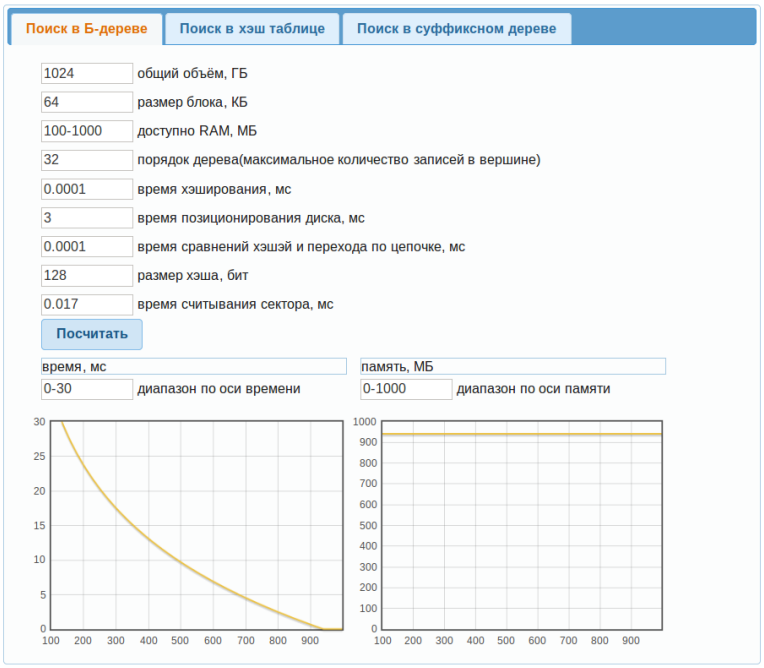
Стоит пояснить пункт, характерный для этой области, но носящий, по большей части, рекламный характер: степень дедупликации — это величина, равная объёму данных в хранилище до дедупликации, делённому на объём данных в хранилище после. Вопрос оценки качества дедупликации тоже ещё не изучен, а главным критерием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

1.5. Обзор “калькуляторов дедупликации”

Существует ряд “рекламных” калькуляторов для оценки степени дедупликации, например
http://www.datadomain.com/calc/index.html

http://eval.symantec.com/flashdemos/products/netbackup/dedupe_calc2/

Летом 2011 года в рамках летней школы Кафедры Системного Программирования был написан калькулятор для теоретической оценки скорости структур данных, которые можно использовать для дедупликации.



2. Постановка задачи

2.1. Формулировка

Разработать инструмент для оценки эффективности дедупликации данных посредством цепочек преобразований.

Будем называть преобразованием обратимое отображение из потока байт в поток байт, а цепочкой преобразований - композицию преобразований. Существует много изученных преобразовний, которые используются в задачах кодирования данных.

На вход инструменту подаются данные и последовательность преобразований, на выходе получается информация об исходном объёме данных, о конечном объёме данных, о распределении данных по типам и о параметрах входного алгоритма.

2.2. Актуальность

Для дедупликации и архивации существет множество различных алгоритмов, которые необходимо сравнивать между собой по многим критериям, в том числе, по количеству выигранного места.

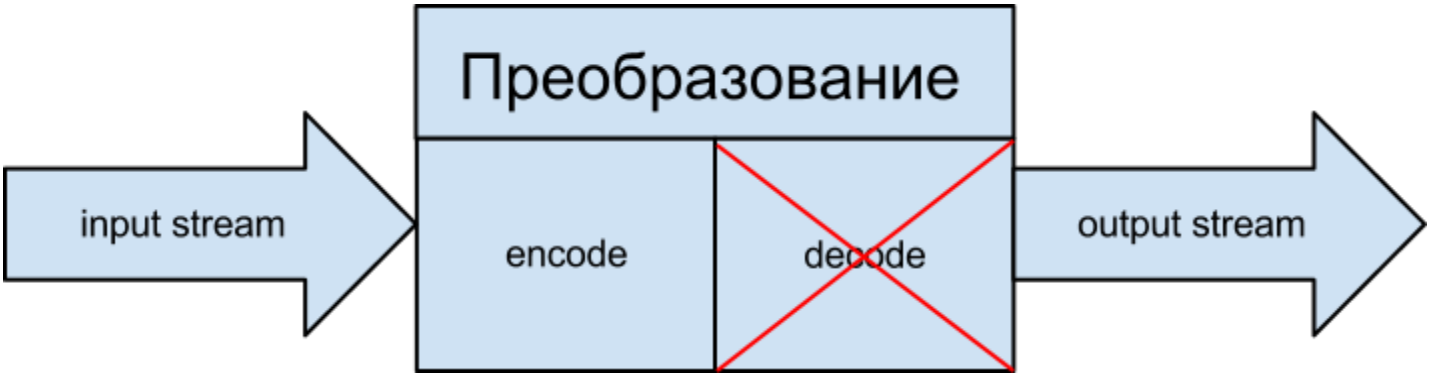
Как было показано, существует несколько решений по оценке дедупликации. Их существенным недостатком является то, что они измеряют эффективоность дедупликации только для фиксированного алгоритма в зависимости от типа данных. Из-за этого их нельзя использовать для выбора наилучшей по экономии места стратегии дедупликации.

3. Решение

Разработка инструмента, который позволял бы сравнивать по dedupe ratio разные алгоритмы между собой.

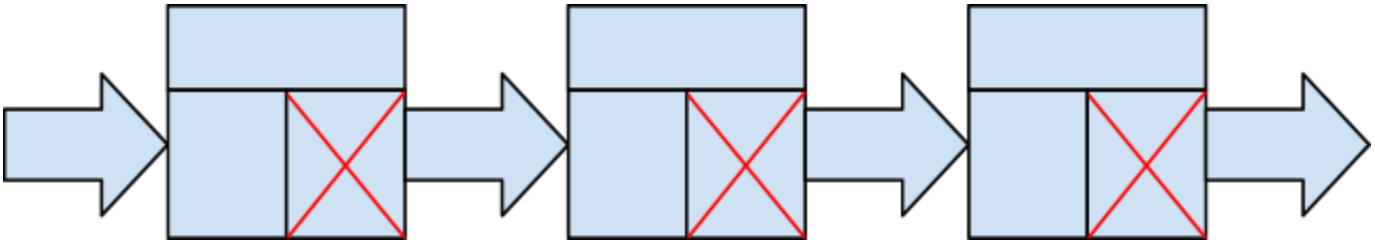
3.1. Архитектура

Как выглядит пребразование:



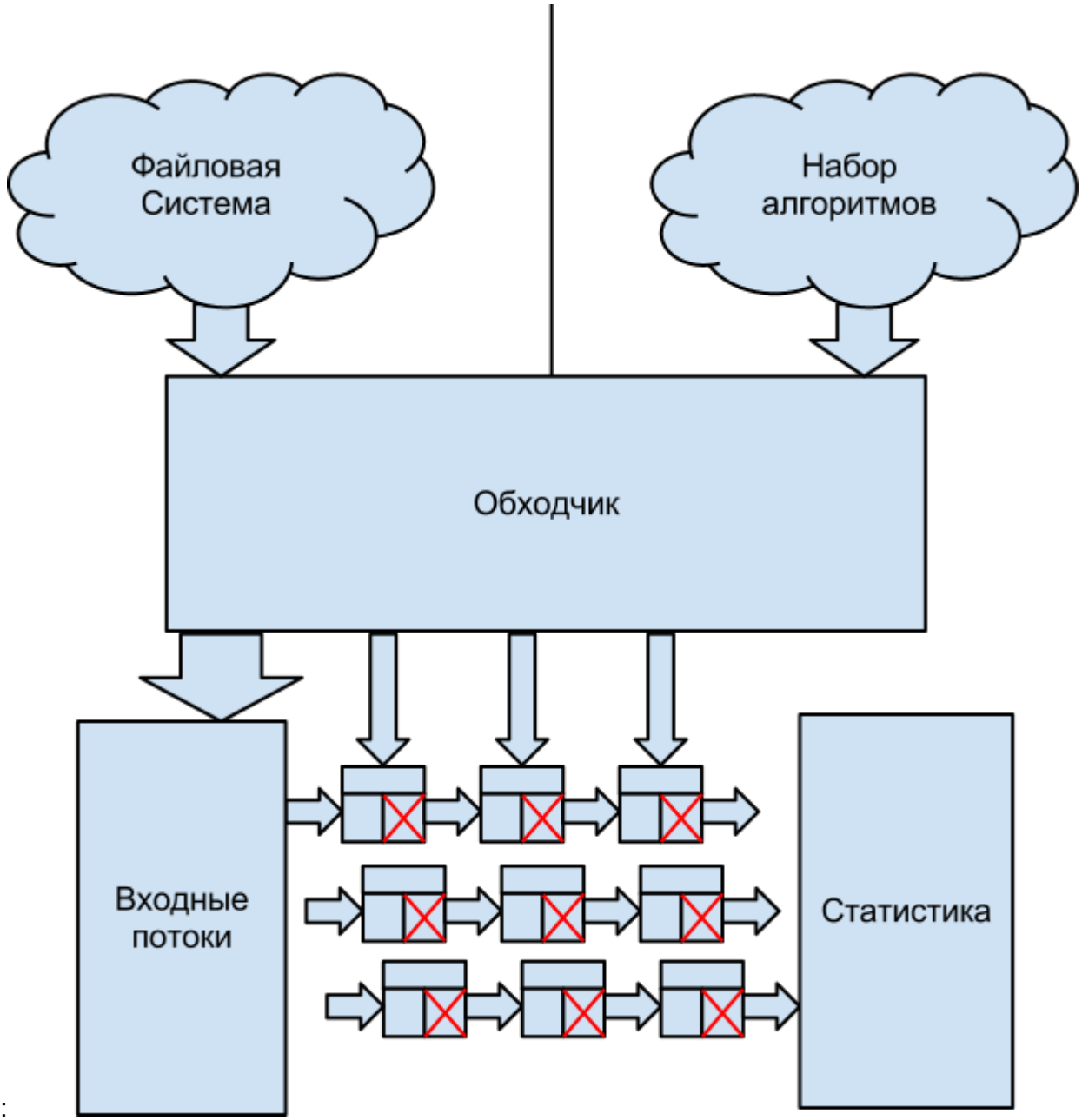
input stream - входной поток байт
output stream - преобразованный поток байт
encode - прямое преобразование, которое из любого входного потока получает его код.
decode - обратное преобразование, которое из корректного кода получает исходный поток; для оценки dedup ratio обратное преобразование не нужно.

Как выглядит цепочка преобразований:



То есть выход каждого преобразования в цепочке без изменений передаётся на вход следующему.

Как выглядит всё



вместе:

Обходчик - модуль, котрый всё связывет. Из заданных преобразований он формирует одну или несколько цепочек и передаёт им на вход потоки, сформированные из файлов файловой системы.

3.2. Используемые технологии

Для реализации был выбран язык Python. Выбор был обусловлен простотой разработки.

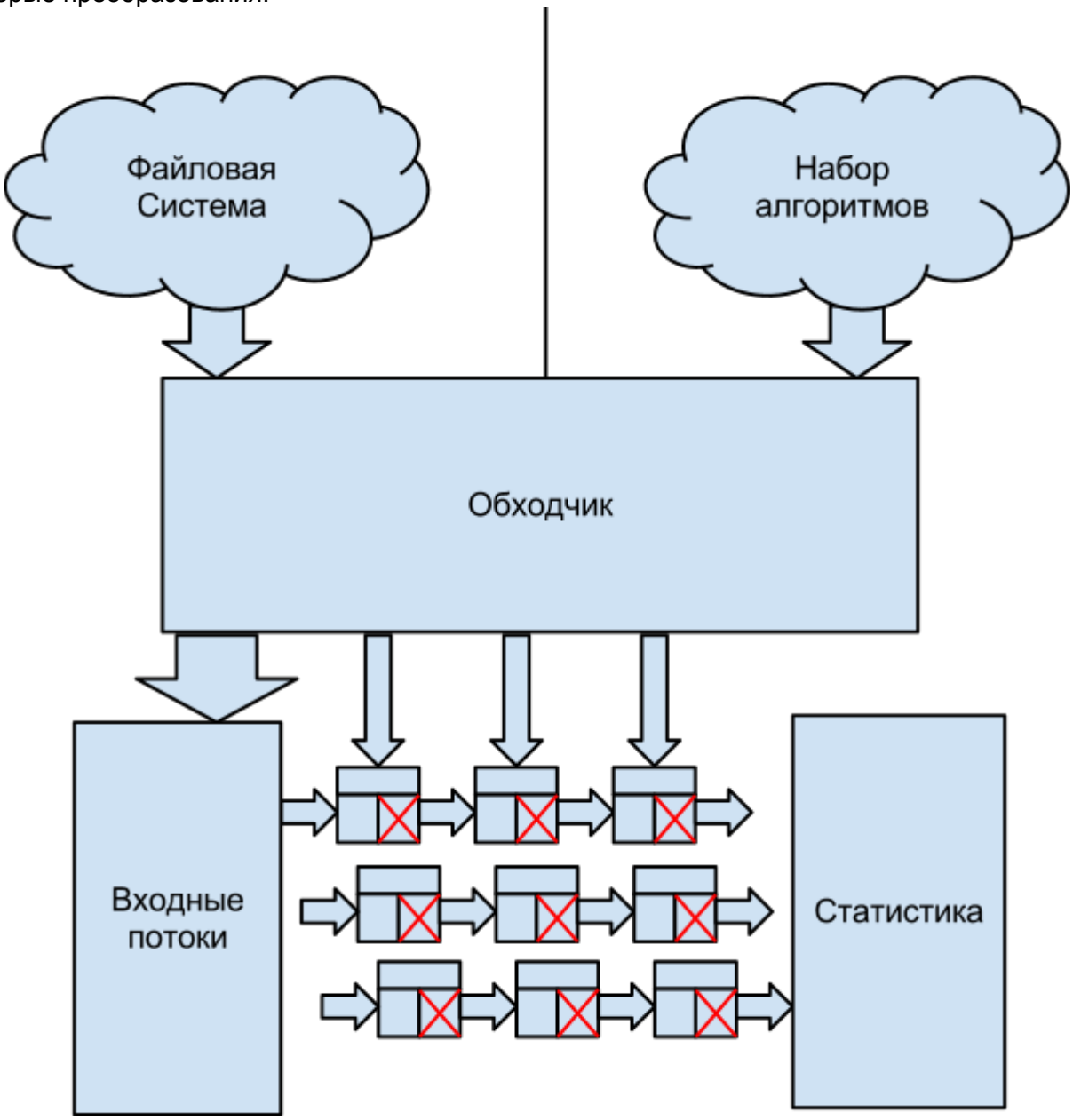
3.3. Трудности

Возникли трудности с использованием уже готовых алгоритмов, так как они не поддерживают необхоимый интерфейс. Например необходимо чтобы преобразоваиня умели работать не только с потоками байт, но и с потоками блоков произвольного размера

4. Результаты

4.1. Архитектура инструментального средства

Разработан прототип на языке Python средства измерения дедупликации. Реализованны некоторые преобразования.



4.2. Оценка без фактического сжатия

Средство было проверенно на различных цепочках и с помошью него были сделаны какие-то выводы.

5. Приложения

5.1. Описание алгоритмов

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.

Интервальное кодирование

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

Кодирование длин серий

Простой алгоритм сжатия данных, заменяющий последовательность из n одинаковых символов на число n и символ. Эффективен только для некоторых типов данных, но при некоторой предварительо обработке может применятся в более широком диапазоне случаев.

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.

LZ78

Алгоритм словарного сжатия, модификация LZ77, в котором исползуется статический словарь, содержимое которого зависит от всего просмотренного сообщения, а не только от содержимого скользящего окна.

Преобразование Барроуза — Уилера

Преобразование сообщения, не являющееся сжатием, но позволяющее впоследствии более эффективно применять такие алгоритмы как LZ78/77 или Run-length encoding. Оно обратимым образом переставляет символы сообщения так, что одинаковые символы идут подряд.

6. Литература

1. Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. Методы сжатия данных. Москва, ДИАЛОГ-МИФИ, 2003.

2. Christopher W. Fraser. An Instruction for Direct Interpretation of LZ77-compressed Programs. September 2002.