Министерство образования и науки Российской Федерации

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

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

РАЗРАБОТКА МЕТОДОВ ХРАНЕНИЯ ИЕРАРХИЧЕСКИ СТРУКТУРИРОВАННЫХ ДАННЫХ С ВЕРСИОНИРОВАНИЕМ В ДОКУМЕНТООРИЕНТИРОВАННОМ ХРАНИЛИЩЕ

Выпускная квалификационная работа
(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:
студент 376 группы _____ Неганов
Алексей Михайлович

Научный руководитель:
к.ф.-м.н. _____ Коротаев
Кирилл Сергеевич

Москва 2017

Содержание

Введение **4**

1 Постановка задачи **6**

2 Обзор источников **7**

2.1 Модель списка смежных вершин 7

2.2 Модель вложенных множеств 8

2.3 Модель материализованных путей 8

3 Методика решения задачи **10**

3.1 Предположения и допущения 10

3.2 Тестовые данные 10

3.3 Запросы к системе 11

3.4 Используемые технические и программные средства 12

3.5 Методика проведения измерений 13

4 Моделирование иерархических связей **14**

4.1 Необходимая терминология 14

4.2 Использование иерархического токенизатора 15

5 Моделирование временных срезов **18**

5.1 Выбор схемы версионирования 18

5.2 Модель, использующая документы-версии 19

5.3 Модели, использующие связь «один ко многим» 19

6 Программная реализация **23**

6.1 Архитектура приложения 23

6.2 Используемые оптимизации 24

7 Результаты **26**

7.1 Проведённые серии экспериментов 26

7.2 Времена исполнения запросов для модели документов-версий . . 27

7.3 Времена исполнения запросов для модели с массивом номеров срезов 36

7.4 Времена исполнения запросов для модели родитель-ребёнок . . . 42

7.5 Размер индекса 47
7.6 Сравнение моделей 47

8 Выводы **49**

Список литературы **51**

Приложение А. Описания индексов в формате JSON **52**

Приложение Б. Запросы к системе на языке Elasticsearch Query DSL **58**

Введение

В настоящее время происходит взрывной рост объёма хранимых и анализируемых данных, используемых для разнообразных целей. В связи с этим приобретают остроту проблемы организации полнотекстового поиска и аналитики в режиме реального времени. Важными требованиями к возможным решениям являются линейная масштабируемость, гибкость схемы данных, возможность развёртывания на распределённой системе. Широкое применение в промышленности получили *документоориентированные поисковые системы*, сочетающие вышеперечисленные свойства с удобством применения в информационном поиске.

Документоориентированная модель не предполагает задания жёстких связей между элементами индексируемой коллекции. В результате проблема применения документоориентированных поисковых систем к данным, имеющим иерархическую организацию, является нетривиальной.

Данная работа представляет собой анализ возможных способов использования документоориентированных баз данных и поисковых систем для решения задач индексации и поиска по иерархически структурированным данным, таким как файловая система, а также пример построения такого поискового индекса на основе программного продукта Elasticsearch с программной реализацией этого решения. Предложено три способа организации версионирования на базе поисковой системы и произведено сравнение производительности важнейших операций и размера поискового индекса для этих способов.

В Интернете можно найти обзорные сведения по использованию документоориентированной модели для иерархических данных ([1]). К сожалению, такие обзоры не содержат подробного анализа приводимых алгоритмов применительно к данной модели, ограничиваясь ссылками на соответствующие исследования для реляционных БД. Существующие работы, посвящённые практическому применению документоориентированных поисковых систем, в частности Elasticsearch, для поиска по иерархическим данным, такие как [2], посвящены почти исключительно деталям программной реализации индексирования структурированных текстовых документов, и их результаты не могут быть механически перенесены на произвольные иерархические данные. Работы, исследующие применение NoSQL парадигмы для задач

версионирования, такие как [3], сфокусированы на графовых БД.

Данное исследование призвано заполнить этот пробел и предложить способы решения задачи применения документоориентированной поисковой системы для поиска по иерархической системе с версионированием.

Актуальным приложением этой работы является система архивации файлов. Результаты данного исследования используются в реализации «каталога данных» продукта Acronis Backup Advanced 12 (компонент Acronis Storage Node).

1 Постановка задачи

Распространённое определение иерархической модели данных состоит в том, что данные организованы древовидным образом. Без потери общности его можно переформулировать так: данные распределены по системе вложенных коллекций.

Организация поиска по данным с помощью документоориентированной системы поиска заключается в том, что данным должен быть сопоставлен набор описывающих их независимых *документов* определённого формата. Эти документы и будут индексироваться и анализироваться системой.

Организация поиска по различным типам иерархически структурированных данных будет отличаться способами построения описывающего данные набора документов, но этот последний будет во всех случаях иметь ряд общих свойств, и, таким образом, рассмотрение некоего конкретного типа данных не сузит общность теоретических выводов. При этом такой переход существенно упростит дальнейшее изложение и приблизит его к вопросам практического применения предлагаемых решений.

В силу тех же причин, а также из-за желания пользоваться единым источником технических терминов, предлагаемые решения будут основаны на конкретной системе документоориентированного поиска — Elasticsearch, т. е. той, на которой основаны программные реализации этих решений.

Цель работы — проанализировать способы использования документоориентированной модели для поиска по иерархически структурированным данным с версионированием.

Из цели работы вытекают следующие **задачи:**

1 Рассмотреть известные алгоритмы описания иерархических структур применительно к документоориентированным хранилищам

2 Описать способы производить поиск по совокупности версий текстовых файлов с учётом их иерархического положения на файловых системах различных компьютеров и версий во времени

3 Программно реализовать эти способы и сравнить их фактическую производительность и масштабируемость

2 Обзор источников

Проблема описания древовидных структур хорошо изучена для реляционных БД ([4], [5]). Хотя результаты этих исследований и нельзя механически перенести на документоориентированные БД и поисковые системы, но тем не менее эти работы вводят основные алгоритмы описания иерархических структур путём введения отношений между данными.

Практические руководства по использованию конкретных программных продуктов ([1]) содержат отрывочную информацию по эффективности применения введённых для реляционных БД алгоритмов применительно к документоориентированным системам.

Руководство [6] содержит подробное изложение возможных способов описания связей между документами средствами системы, с описанием принципиальных ограничений и возможных путей потери производительности.

2.1 Модель списка смежных вершин

Модель списка смежных вершин задаёт иерархию путём хранения связей между родительскими и дочерними узлами. Эта модель представляет собой традиционный в информатике способ описания деревьев и древовидных структур как ациклических неориентированных графов.

Для документоориентированных поисковых систем и БД сохраняется свойство, сформулированное в ([4]) для реляционных БД: эта модель обладает существенными достоинствами для записи — в дерево легко вносить изменения, менять местами и удалять узлы.

Тем не менее, она совершенно неприменима для задач поиска. Если в реляционной БД производительность операций чтения низкая, но всё же удовлетворительная, то для документоориентированной поисковой системы она совершенно неприемлема. Дело в том, что в последней поиск осуществляется с помощью обратного индекса (подробнее см. раздел 4.1), когда каждому терму сопоставляется список документов, его содержащих. Применение модели списка смежных вершин означает, что для выяснения факта, принадлежит ли документ тому или иному поддереву (директории) придётся осуществить перебор узлов этого поддерева, т. е. эффективность обратного индекса сводится на нет.

В [1] рассматривается модель «массива предков», которая может быть рассмотрена как развитие модели списка смежных вершин. Эта последняя предполагает хранение массива всех предков узла, что решает вышеописанную проблему. Сравнение модели «массива предков» с предлагаемой в данной работе см. в разделе 4.2.

2.2 Модель вложенных множеств

Эта модель введена в ([4]). Каждый узел идентифицируется, как пара номеров остановок полного обхода дерева. Узел посещается дважды: первый раз при прямом ходе и второй при обратном. Для документоориентированной модели в [1] предлагается в документе, описывающем узел, хранить идентификатор родителя, а также номера первой и второй остановки алгоритма полного обхода на узле.

Модель вложенных множеств хороша для нахождения поддеревьев, но неэффективна для изменения древовидной иерархии. При использовании версионирования последнее обстоятельство несущественно, т. к. каждый срез (см. раздел 5) содержит раз и навсегда зафиксированное состояние данных. Тем не менее, эта модель не лучшим образом подходит для решения поставленной задачи, так как в общем случае, когда узлы иерархического дерева идентифицированы строками, необходимо на этапе индексации узла производить анализ и сортировку имён всех его дочерних узлов. Кроме того, использование модели вложенных множеств означает использование строковых сравнений при поисковом запросе вместо эффективного использования обратного индекса.

2.3 Модель материализованных путей

В модели материализованных путей документ, описывающий узел, хранит в виде строки путь до данного узла. Эта модель обеспечивает большую гибкость, позволяя осуществлять поисковые запросы по иерархии с помощью строковых функций и регулярных выражений. Поиск файла в заданной директории в таком случае будет осуществляться с помощью запросов по префиксу хранимого поля.

Такой подход имеет то преимущество перед предлагаемым в данной работе, что он может быть реализован без использования специальных токенизаторов для гипотетической системы, где таковые отсутствуют.

Алгоритм выполнения такого запроса следующий:

1 Найти терм с заданным префиксом путём итерации по инвертированному индексу

2 Собрать соответствующие ID документов

3 Повторять, пока не достигнут конец индекса

Ясно (и подчёркнуто в документации [6]), что эффективность и масштабируемость подобного подхода крайне низки.

Время выполнения такого запроса — $O(n)$, где n — количество содержащихся в обратном индексе термов с данным префиксом ([6]). Для поиска файла в заданной директории n — количество всех файлов, содержащихся в данной директории с бесконечным уровнем вложенности.

3 Методика решения задачи

3.1 Предположения и допущения

Предполагается, что исходный файлы представляют собой простой текст в кодировке UTF-8 [1]. В случае иной кодировки файлов задача сводится к исходной путём перекодировки; в случае более сложной структуры файла — путём увеличения полей в схеме индекса.

Никаких предположений о характере или языке текста не делается, что означает использование стандартного анализатора.

3.2 Тестовые данные

Как уже говорилось ранее, тестовая модель представляет собой совокупность текстовых файлов, расположенных в системе вложенных каталогов. Так как одной из основных функций системы является полнотекстовый поиск, то для имитации реальной нагрузки имеет смысл искать тексты с достаточно разнообразными термами.

Количество индексируемых данных выбиралось исходя из соображения, что размер поискового индекса в аналогичных системах может составлять величину порядка сотен гигабайт ([7]).

В качестве источника данных был выбран ресурс `http://commoncrawl.org/`[2], содержащий текстовые данные, собранные поисковыми роботами в Интернете в период с апреля 2014 по апрель 2017 года.

Данные содержат большое количество текстовых фрагментов, написанных людьми — пользователями Интернета на разных языках.

В качестве формата входных данных был избран WET[3], который в контексте данной работы не отличается от простого текста.

[1] Unicode Transformation Format, 8-bit — «формат преобразования Юникода, 8-битный» — одна из общепринятых и стандартизированных кодировок текста

[2] CommonCrawl Foundation, 9663 Santa Monica Blvd., #425, Beverly Hills, CA 90210

[3] Web Extracted Text

3.3 Запросы к системе

Для проведения экспериментов было выбрано несколько характерных запросов:

1 Загрузка записей о файлах

2 Выдача файла по его полному пути

3 Поиск файлов, содержащих определённые термы (полнотекстовый поиск)

4 Поиск файлов, содержащихся в определённой директории с бесконечной вложенностью

5 Поиск файлов и директорий, непосредственно содержащихся в определённой директории (листинг содержимого)

6 Поиск файлов по имени

7 Поиск и листинг актуальных версий файлов

8 Поиск и листинг определённых версий файлов

Тексты запросов представлены в разделе Приложение Б.

3.4 Используемые технические и программные средства

Все вычислительные эксперименты проводились на одной технологической платформе.

В качестве поисковой системы выбрана Elasticsearch — наиболее популярная ([8]) система такого рода в мире на момент написания данной работы. Данная система написана на Java и основана на библиотеке Apache Lucene. Предназначенная для развёртывания на распределённой системе, она предоставляет доступ к своим функциям по протоколу HTTP[1].

Используемым в Elasticsearch форматом документов является JSON[2].

Система была настроена согласно указаниям официального технического руководства. Следует отметить отдельно, что в состав виртуального вычислительного кластера было включено более одного виртуального вычислительных узла[3] на индекс, т. е. у системы могла возникнуть необходимость в задании маршрутов для данных и запросов.

Используемая в вычислительных экспериментах программа, обращающаяся к интерфейсу Elasticsearch по HTTP, была написана на C.

Таблица 1: Аппаратные характеристики платформы

Архитектура	x86-64
Частота процессора	2.60 ГГц
Количество процессоров	32
Оперативная память	64234 Мб
L2 кэш	256 Кб

Таблица 2: Используемое программное обеспечение

Поисковая система	Elasticsearch 2.4.5
Операционная система	Linux 3.10.0 (CentOS 7.2.1511)
Java Virtual Machine	OpenJDK 1.8.0
Компилятор C	GCC 4.8.5
Используемые библиотеки	libcurl 7.54.1, OpenSSL 1.0, jsmn

[1] HyperText Transfer Protocol — протокол прикладного уровня передачи данных

[2] англ. JavaScript Object Notation — текстовый формат обмена данными

[3] англ. *shard* ([6])

Таблица 3: Используемые настройки

Java heap size	31553 Мб
Максимальное число файловых дескрипторов	65535
Подкачка страниц виртуальной памяти	отключена
Количество вычислительных узлов на индекс	5

3.5 Методика проведения измерений

Время выполнения операций измерялось с помощью POSIX-утилиты[1] `time`. Так как измеряемые времена много больше одной секунды, предоставляемая данной программой точность измерений вполне удовлетворительна.

Статистика поискового кластера (размер индекса, объём данных сразу после завершения программы) приводится на основе сообщений встроенной диагностики Elasticsearch.

[1]POSIX — англ. portable operating system interface — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой

4 Моделирование иерархических связей

4.1 Необходимая терминология

Ясность дальнейшего изложения требует введения ряда понятий, специфичных для Elasticsearch. Нижеследующие определения приводятся согласно принятым в официальной документации описаниям. Ввиду отсутствия официального русского перевода, в скобках приводятся оригинальные англоязычные термины.

Определение 4.1.1. *Индексом (index)* называется пространство имён, логически обобщающее реальные виртуальные вычислительные узлы и блоки хранения данных документов *(shards)*. С индексом связаны различные настройки и типы документов.

Определение 4.1.2. *Схемой (mapping)* называется описание полей документа. Для поля описывается его тип данных и то, как оно должно индексироваться и храниться. Поисковый движок Lucene не работает со схемами; они нужны для корректного преобразования исходного документа в его внутреннее представление простой плоской структуры.

Определение 4.1.3. *Типом (type)* называется класс документов с именем и *схемой*. Lucene не имеет понятия типа, поэтому каждый проиндексированный документ просто имеет поле `_type`. Исходя из этого, разумно использовать в одном индексе только типы со слабо отличающимися схемами (или связанные каким-то отношением), иначе у каждого документа будет много пустых полей.

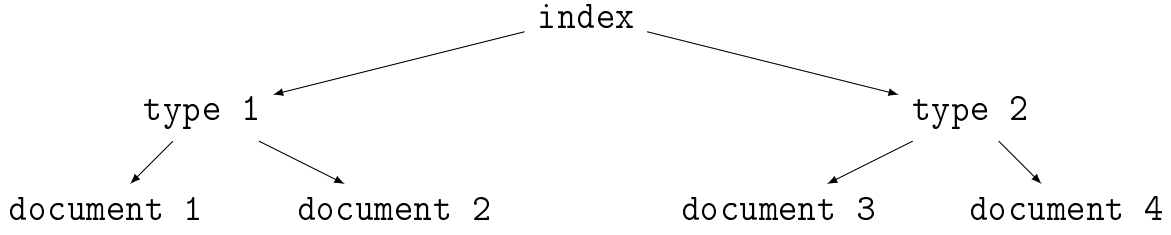

Рис. 1: Логическая структура хранилища

Полнотекстовый поиск в Elasticsearch основан на применении *обратной индексации*, когда для каждого уникального терма составляется список документов, его содержащих.

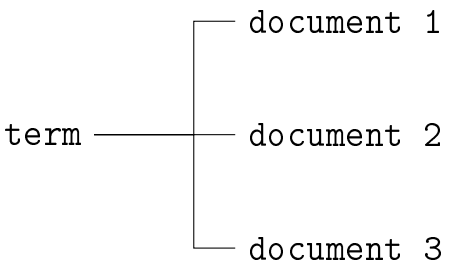

Рис. 2: Обратный индекс

Определение 4.1.4. *Анализ (analysis)* Процесс преобразования исходного текста (содержимого поля) в набор стандартизированных термов для их последующего добавления в обратный индекс. Он состоит из фильтрации символов, токенизации и фильтрации токенов.

В случае, если схема документа задаёт поле как неанализируемое, то поисковые запросы должны проверять точное совпадение термов. В результате такого запроса система либо находит, либо не находит соответствующие документы. Если же поле анализируется, то сравниваются соответствующие результаты анализа, причём система может выдавать в ответ на запрос список документов, ранжированный по точности совпадения.

Анализ обычно применяется для полнотекстового поиска, а неанализируемые поля — для поиска по метаданным.

4.2 Использование иерархического токенизатора

Файловый путь не является неструктурированным текстом: его можно представить в виде совокупности термов — имён вложенных директорий и конечного файла, разделённых специальным символом.

Предложим способ поиска файла с параметром нахождения в заданной директории (с бесконечным возможным уровнем вложенности относительно неё), за время $O(1)$ от количества директорий в пути и за $O(1)$ от количества элементов в директории.

Иерархический токенизатор даёт терм для помещения в обратный индекс на каждый объект соответствующего дерева.

Если при загрузке пути анализируются иерархическим токенизатором, а при поиске задаются как неанализируемые термы, то поиск файлов, содержащихся в некоторой директории, сводится к поиску описывающих

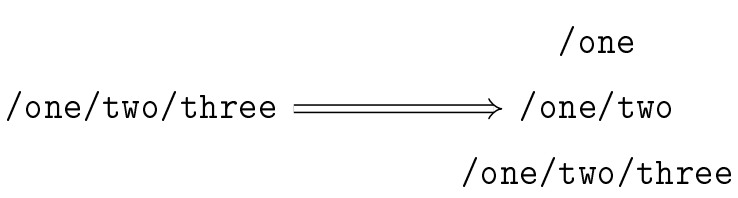

Рис. 3: Схема работы иерархического токенизатора

файлы документов, содержащих терм — путь до заданной директории, что удовлетворяет заявленным требованиям ввиду наличия обратного индекса, сопоставляющего каждому терму список содержащих его документов.

Elasticsearch позволяет индексировать одно и то же поле несколько раз с разными настройками анализа. Для поиска по имени файла путь можно индексировать второй раз с токенизацией в обратном порядке.

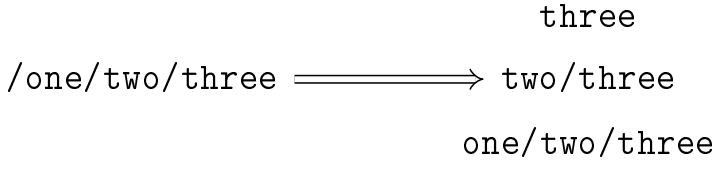

Рис. 4: Обратная токенизация

Для того, чтобы искать документы, содержащиеся непосредственно (т. е. с уровнем вложенности 1) в некоторой директории, необходимо индексировать путь ещё раз, как неанализируемое поле.

Хотя формально информация о директориях уже заключена в пути, единственный способ эффективно выдавать информацию о них при листинге — хранить специальные директорные документы для каждой директории. Кроме того, такой подход позволяет индексировать метаданные директорий. Из соображений эффективности (чтобы не получить большое количество документов с большинством пустых полей) разумно использовать не директорный тип, а отдельный индекс. Возможны поисковые запросы по нескольким индексам одновременно.

Отметим, что анализирование пути иерархическим токенизатором позволяет реализовать операции над директориями (например, удаление из индекса директории с содержимым) без рекурсивного обхода: можно получать идентификаторы всех потомков данной директории за один запрос.

Таким образом, схема описывающего файл документа должен содержать, помимо поля с анализируемым текстом, три поля, описывающих путь: для

прямой иерархической токенизации, для обратной и неанализируемое поле для листинга. Полные описания схем в формате JSON представлены в разделе Приложение А.

Заметим, что получившееся решение сущностно подобно рассматриваемой в [1] модели «массив предков» *(array of ancestors)*, но имеет также и существенное отличие. Модель «массив предков» подразумевает хранение массива идентификаторов всех вложенных директорий; для получения этих идентификаторов пришлось бы совершать поиск и, таким образом, время индексации имело бы асимптотику $O(n)$, где n — количество директорий в пути. Предложенный же способ не страдает от этого недостатка, т. к. помещает в документ эквивалентную информацию, работая в путём как с простым текстом.

5 Моделирование временных срезов

5.1 Выбор схемы версионирования

Целью и смыслом версионирования является учёт изменений данных на протяжении некоторого отрезка времени. Ясно, что учёт версий для каждого файла независимо от прочих нецелесообразен в случае, если предполагается обращение с данными как с единым целым, что включает в себя поиск и разнообразную аналитику и определено контекстом данной работы.

Определение 5.1.1. *Схема версионирования* — логическое представление совокупности состояний данных, связанных с определёнными отметками во времени.

Понятия, связанные с версионированием, хорошо развиты и определены в области резервного копирования. Дальнейшие определения данного раздела приводятся в соответствии с [9].

Определение 5.1.2. *Срез (slice)* — реализация представления состояния данных в определённый момент времени.

Определение 5.1.3. *Полным срезом (full slice)* называется срез состояния данных, содержащий полную информацию о данных в заданный момент времени. Полные срезы содержат описания всех актуальных на определённый момент времени версий файлов.

При обновлении изменяется состояние лишь части файлов. Полные срезы содержат большое количество избыточной информации. В целях оптимизации вводятся специальные виды срезов, которые позволяют получить информацию о состоянии данных в заданный момент времени только при слиянии с информацией из других срезов. Хотя существует несколько стратегий создания таких неполных срезов (инкрементальная, дифференциальная, декрементальная) в контексте данной работы разница между ними несущественна и для целей тестирования может быть избрана любая из них.

Определение 5.1.4. *Инкрементальным срезом (full slice)* называется срез состояния данных, только обновлённые версии по сравнению с предыдущим срезом.

Удобно иметь возможность искать по актуальному состоянию системы, даже если соответствующего полного среза не было создано.

5.2 Модель, использующая документы-версии

Добавим к описывающему файл документу два неанализируемых поля: номер среза, в состав которого добавлен файл, и булево значение актуальности. Для инкрементальных срезов добавим булев атрибут `deleted`, чтобы с помощью специальных документов отмечать удаление файла в соответствующем срезе.

Тогда выполнение поисковых запросов можно описать следующим образом:

1 Поиск по заданному полному срезу: поиск с параметром — номером полного среза

2 Поиск по актуальному состоянию данных: поиск с параметром — актуальностью ИСТИНА

3 Поиск по заданному инкрементальному срезу: слияние результатов поиска по всем срезам от последнего предшествующего полного среза, где в случае наличия документа с некоторым путём в более чем одном срезе выбирается наиболее свежая версия.

Недостатки подхода:

1 Если при обновлении не произошло фактического изменения содержания файла (такие ситуации будут, в частности, возникать при создании full slice), то произойдёт копирование информации в индексе.

2 При каждом обновлении надо производить переиндексацию всех актуальных файлов, чтобы сбросить у них атрибут актуальности (документы являются атомарными сущностями, и при обновлении фактически происходит выгрузка документа из индекса и загрузка обновлённого).

5.3 Модели, использующие связь «один ко многим»

Представим некое состояние файла в виде главного объекта, содержащего информацию о файле как таковом, и совокупности подчинённых объектов,

содержащего информацию о срезах, в которых файл имеет это состояние. Такой подход позволяет не дублировать неизменённые файлы между срезами и отказаться от атрибута актуальности. Понятие инкрементального среза при этом становится бессмысленным.

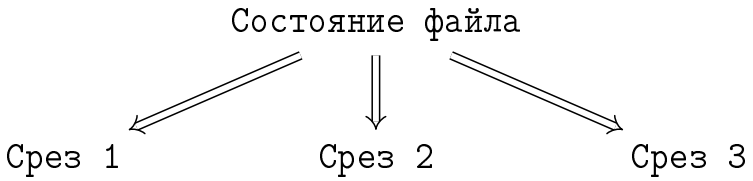

Рис. 5: Отношение между состоянием файла и соответствующими ему срезами

Для документоориентированных систем характерна плоская модель данных: индекс — это совокупность независимых документов, каждый из которых должен содержать всю информацию, необходимую для поиска. Изменение одного отдельно взятого документа удовлетворяет требованиям ACID[1] ([6]), но этого нельзя сказать об изменении некоторого множества документов. Подобная архитектура позволяет добиться быстрых и неблокирующих операций, а также относительной лёгкости распределения данных по компьютерам распределённой системы.

Тем не менее, с определёнными ограничениями всё же можно связать один объект с другим. Elasticsearch имеет две встроенных реализации отношения «один ко многим» — так называемые модели вложенных объектов *(nested objects)* и родитель-ребёнок *(parent-child)*.

Модель вложенных объектов позволяет индексировать части документа как отдельные скрытые документы. Эта модель рекомендуется для использования в случае наличия небольшого ограниченного количества подчинённых объектов. Основной недостаток этой модели — при добавлении, удалении или обновлении подчинённого объекта происходит переиндексация всего документа.

Так как в подчинённом объекте находится только один элемент (номер среза), то использование модели вложенных объектов можно эффективно заменить на использование массива номеров срезов как элемента схемы. В этом случае можно избавиться от накладных расходов и убрать ограничение на количество подчинённых объектов. Массивы (многозначные поля) хорошо

[1]англ. Atomicity, Consistency, Isolation, Durability — Атомарность, Согласованность, Изолированность, Устойчивость

поддерживаются, так как они имеют ту же природу, что и наборы термов, получаемые в результате анализа некоего текстового поля.

Выполнение поисковых запросов для модели массива номеров можно описать так:

1 Поиск по заданному срезу: поиск документов, имеющих номер среза

2 Поиск по актуальному состоянию данных: поиск документов, имеющих номер последнего среза

Модель родитель-ребёнок позволяет связывать отдельные документы между собой. Она имеет следующие свойства:

1 Родительский документ может быть обновлён без переиндексации дочерних

2 Добавление, удаление или изменение дочернего документа не затрагивает родительского и других дочерних

3 Можно искать родительские документы по наличию определённых дочерних и наоборот

Система накладывает органичение на расположение связанных объектов: родительский документ и его дочерние должны находиться на одном виртуальном вычислительном узле *(shard)*. Отношения поддерживаются с помощью глобального отображения в оперативной памяти, что означает сравнительно бóльший расход памяти приложением. При поиске происходит, если это необходимо, слияние документов. Каждое вносимое в индекс изменение будет нести за собой обновление глобальных ординалов, используемых для ускорения слияний. Согласно документации ([6]) в отдельных случаях поиск при использовании модели родитель-ребёнок может быть в 5 – 10 раз медленнее по сравнению с моделью вложенных объектов. Тем не менее, для случая сравнительно небольшого количества родительских документов по сравнению с дочерними эта модель объявляется предпочтительной.

Выполнение поисковых запросов для модели родитель-ребёнок можно описать следующим образом:

1 Поиск по заданному срезу: поиск родителя с ребёнком, имеющим номер среза

2 Поиск по актуальному состоянию данных: поиск родителя с ребёнком, имеющим номер последнего среза

Характеристики моделей сведены в нижеследующей таблице.

Таблица 4: Сравнительные характеристики моделей версионирования

Модель	Дедупликация	Быстрый поиск	Переиндексация неизменивших-ся файлов	Время выполнения запроса от числа инкрементов
Документы-версии	нет	да	да	$O(n)$
Родитель-ребёнок	да	нет	нет	$O(1)$
Массив номеров	да	да	да	$O(1)$

Отсюда видно, что модель документов-версий подходит только для быстро меняющихся данных (мало совпадений между срезами), в противном случае, если важна скорость поиска, то оптимальна модель с массивом номеров, а если важна скорость индексации, то возможно лучше подойдёт модель родитель-ребёнок. Применимость этих теоретических выводов на практике см. в разделах 7 и 8.

6 Программная реализация

6.1 Архитектура приложения

Тестовое приложение написанно на языке C. Для работы с Elasticsearch REST[1] API[2] использовалась библиотека libcurl. Для десериализации JSON использовалась библиотека jsmn.

Возможна развёртка приложения в виде совокупности сервисов, осуществляющих сериализацию и загрузку данных с компьютеров-источников и сервера-посредника для обработки административных и поисковых запросов. Поисковые запросы могут осуществляться и напрямую через Elasticsearch API; роль посредника состоит в том, чтобы применять рассмотренные в этой работе шаблоны и составлять оптимальные для фиксированных (см. раздел 3.3) сценариев запросы на языке Elasticsearch Query DSL[3]

Для определения идентичности файлов используется путь и временная метка последней модификации. При загрузке каждого файла происходит поиск идентичных ему.

Для работы с иерархией используется рассмотренный в разделе 4.2 иерархический токенизатор. Выбор модели версионирования из числа рассмотренных в разделе 5 осуществляется при создании индекса.

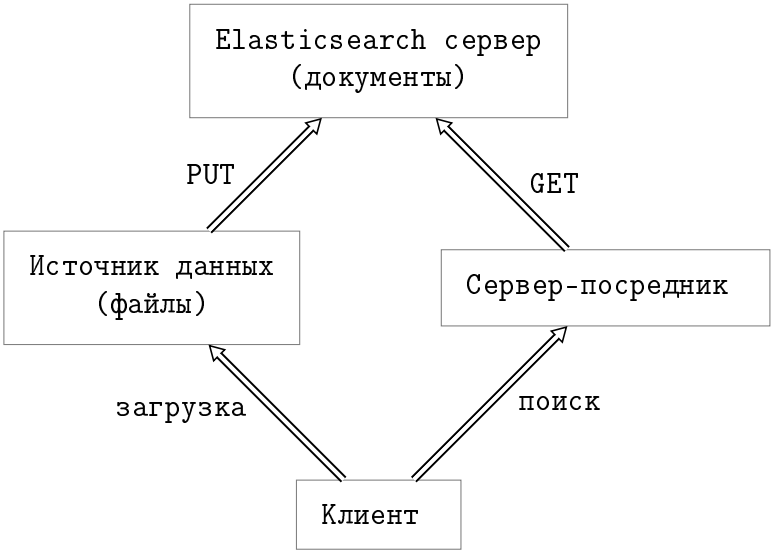

Рис. 6: Исполнители запросов

[1]REST (сокр. от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети

[2]API (англ. application programming interface) — программный интерфейс приложения

[3]DSL (англ. Domain-specific language) — предметно-ориентированный язык

Для поиска создаются два индекса: файловый и директорный. Кроме того, в силу того что не используется никакая другая база данных, создаётся вспомогательный индекс срезов для хранения информации о номере последнего среза и управления срезами; все поля в этом случае неиндексируемые.

6.2 Используемые оптимизации

Существенными являются детали реализации загрузки файлов, т. к. для таких запросов кроме обращения к Elasticsearch серверу происходит нетривиальное сопоставление данным описывающего их набора документов (см. раздел 1). Таким образом, для измерения скорости загрузки файлов используемое для тестов приложение является фактором, которым нельзя пренебречь.

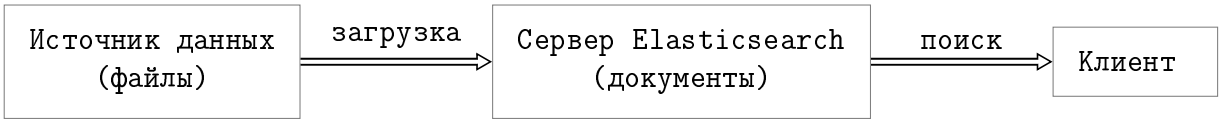

Рис. 7: Перемещение данных в ходе использования системы

Для файла описанного выше (раздел 3.1) вида преобразование содержимого в JSON формат сводится к экранированию символов и расстановке специальных знаков (скобок, кавычек, двоеточий и т. д.) Такое преобразование может быть осуществлено в один проход без синтаксического анализа входного файла. В контексте задачи используется небольшое фиксированное количество видов запросов к Elasticsearch серверу. Исходя из этого, сериализация запросов осуществляется «вручную», без использования сторонних библиотек, с целью упрощения и оптимизации.

Для того, чтобы эффективно определять, был ли загружен данный файл в предшествующих срезах, нужно генерировать его ID, не полагаясь на автогенерацию Elasticsearch. Файл определяется своим месторасположением с точностью до версии. Для построения ID берётся хэш от сконкатенированной строки «машина + путь + слайс создания». В качестве хэш-функции используется SHA1, как достаточно устойчивая к коллизиям.

Elasticsearch позволяет группировать запросы на поиск, индексацию, создание, обновление и удаление документов, что может существенно ускорить такие операции. Оптимальный размер группового *(bulk)* запроса — примерно 10 – 15

Мб. Наличие групповых запросов особенно важно для директорных документов небольшого размера. Кроме того, в варианте модели версионирования с документами-версиями группируются множественные запросы на поиск (путь + актуальность) для обновления.

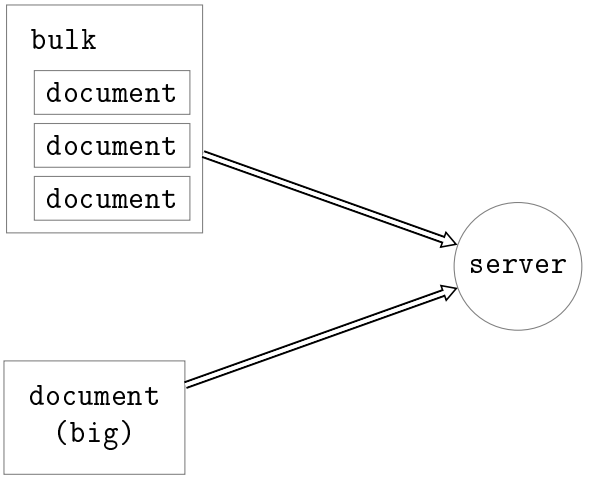

Рис. 8: Группировка запросов на индексацию

В данной реализации используется функция «загрузки директории», которая индексирует все файлы, содержащиеся в данной директории; при этом директорные документы и файловые документы малого размера группируются для загрузки на Elasticsearch-сервер, а большие файлы обрабатываются отдельно.

7 Результаты

7.1 Проведённые серии экспериментов

Текстовые данные, описанные в разделе 3.2, были разделены на файлы по 1000 строк в каждом. Эти файлы были распределены по 125 каталогам, которые были вложены в 25 каталогов по 5 подкаталогов, которые были вложены в 5 каталогов по 5 подкаталогов.

Для каждой контрольной точки создавалось два (полных) среза с идентичными данными. Такой подход позволяет хорошо сравнить эффекты от дедупликации и переиндексации. Для каждой новой точки создавались новые срезы; таким образом, увеличение объёма данных происходило с увеличением количества срезов.

Для всех типов поиска приводятся средние результаты за 1000 запросов. На графиках для каждой контрольной точки вместе со средним значением показано среднеквадратичное отклонение.

Размер страницы поисковой выдачи — 3000. При поиске всегда получалась только первая страница. Отсюда следует, что на графиках времена полнотекстового поиска должны быть примерно в $\sim 10^3$ раз больше времён выдачи единственного файла.

Пометкой «64» и вертикальной пунктирной линией обозначена величина обрабатываемых данных в 64 Гб, что соответствует объёму оперативной памяти на машине.

Таблица 5: Контрольные точки измерений

Количество файлов	97 368	486 839	973 724	1 460 003	1 946 401
Объём в гигабайтах	3,921	19,583	39,162	58,722	78,266
Количество файлов	2 432 044	4 643 288	6 580 739	8 277 193	9 974 312
Объём в гигабайтах	97,842	186,788	264,697	332,993	401,462

7.2 Времена исполнения запросов для модели документов-версий

Таблица 6: Времена загрузки для модели документов-версий

Объём в гигабайтах	3,921	19,583	39,162	58,722	78,266
Время в секундах (1 срез)	926	5077	10 157	15 394	20 721
Время в секундах (2 срез)	345	4 448	9 754	15 120	20 459
Объём в гигабайтах	97,842	186,788	264,697	332,993	401,462
Время в секундах (1 срез)	26 056	50 535	71 497	90 042	108 633
Время в секундах (2 срез)	25 836	50 285	71 297	89 851	108 402

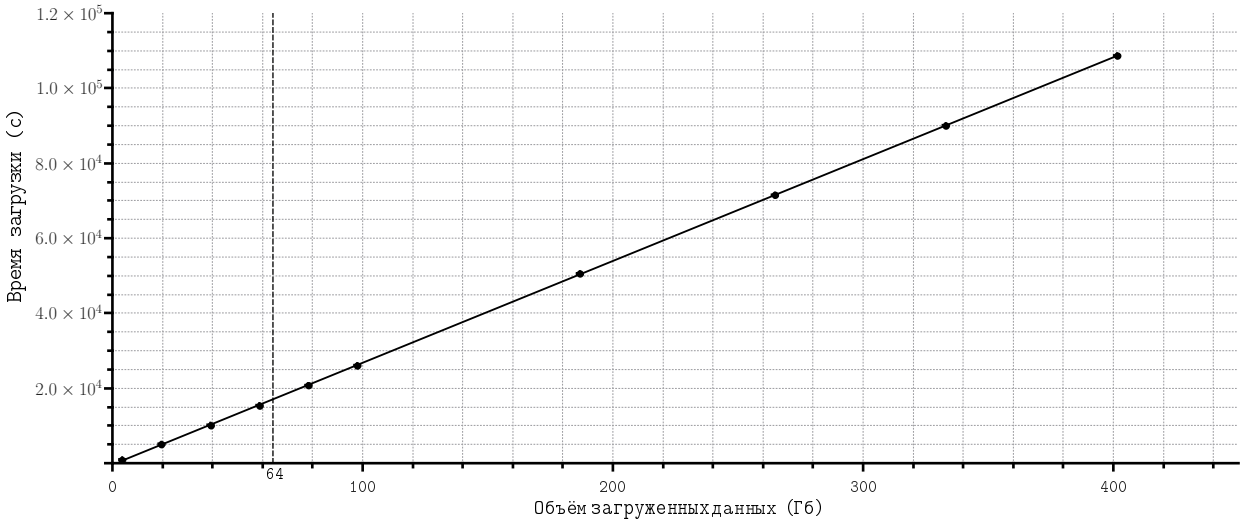

Рис. 9: Зависимость времени загрузки от объёма данных для модели документов-версий

Видно, что время загрузки растёт линейно, т. е. время удельное время загрузки документа с хорошей точностью константно.

На следующем графике линией обозначено приближение функцией экспоненциального затухания. Такой результат можно объяснить наличием кэширования в памяти.

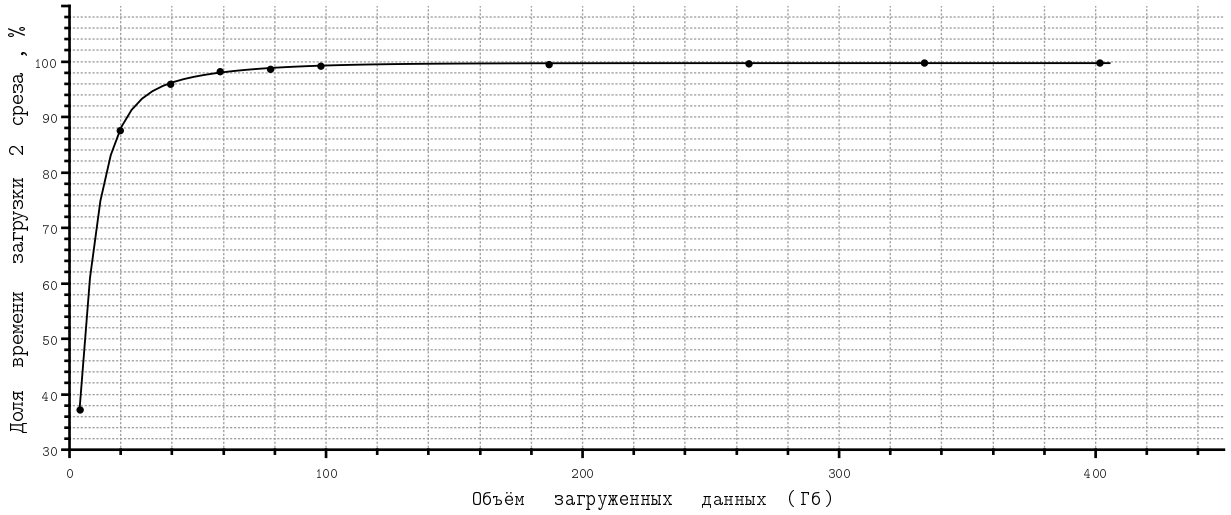

Рис. 10: Доля времени загрузки второго среза по сравнению с первым для модели документов-версий

Следующий ряд экспериментов показывает, что среднее время выполнения основных поисковых поисковых запросов для рассматриваемой модели константно в пределах погрешности измерения.

Таблица 7: Средние времена поиска файла по полному пути для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	11,6	12,1	11,7	11,2	11,6	11,8	11,2

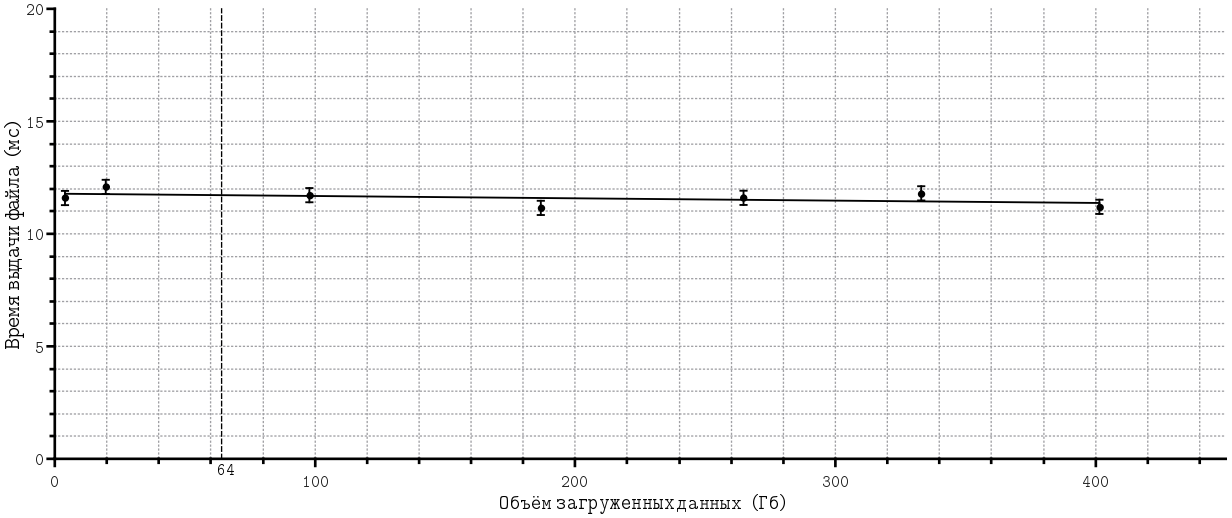

Рис. 11: Зависимость среднего времени поиска файла по полному пути от объёма данных для модели документов-версий

Таблица 8: Средние времена полнотекстового поиска для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в секундах	1,70	1,54	1,68	1,50	1,57	1,66	1,53

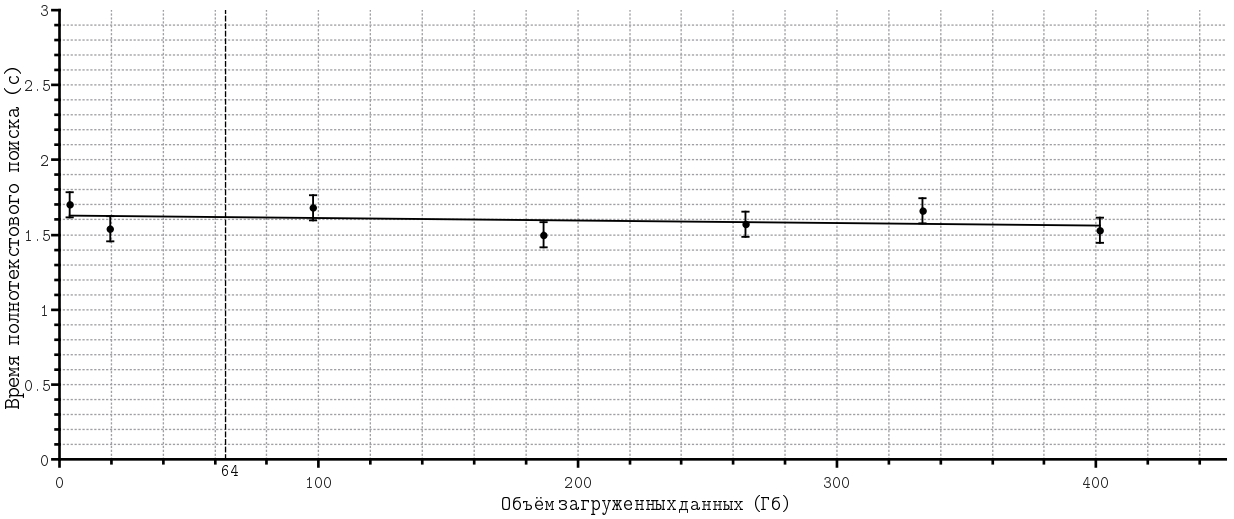

Рис. 12: Зависимость среднего времени полнотекстового поиска от объёма данных для модели документов-версий

Таблица 9: Средние времена полнотекстового поиска с указанием директории для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в секундах	2,56	2,57	2,54	2,30	2,50	2,40	2,25

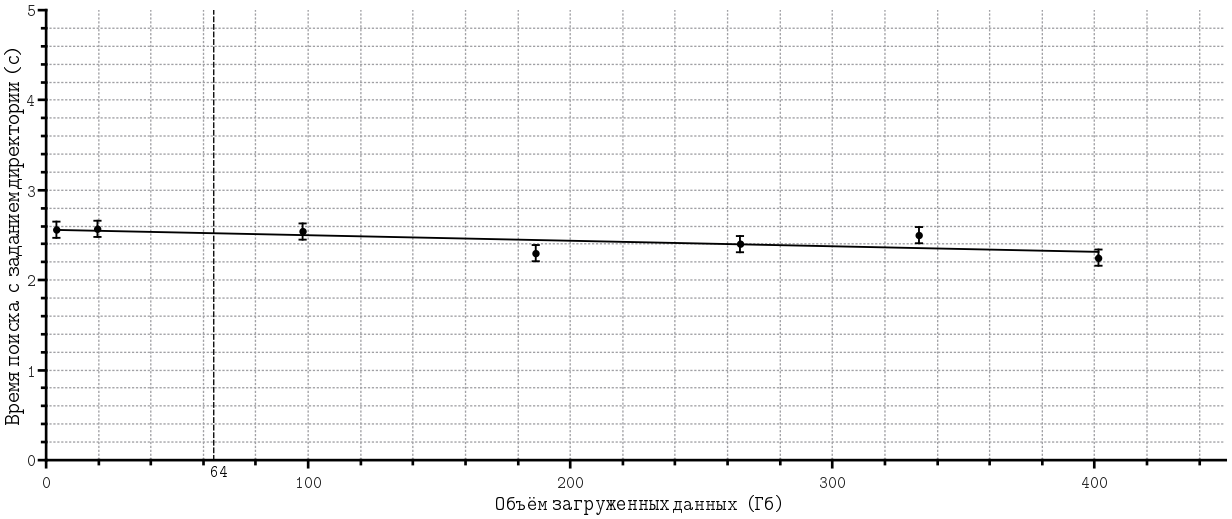

Рис. 13: Зависимость среднего времени полнотекстового поиска с с указанием директории от объёма данных для модели документов-версий

Видно, что время поиска с указанием директории примерно в 1.5 раза больше, чем время простого поиска.

Таблица 10: Средние времена листинга директории для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в секундах	1,20	1,10	1,23	1,10	1,13	1,09	1,09

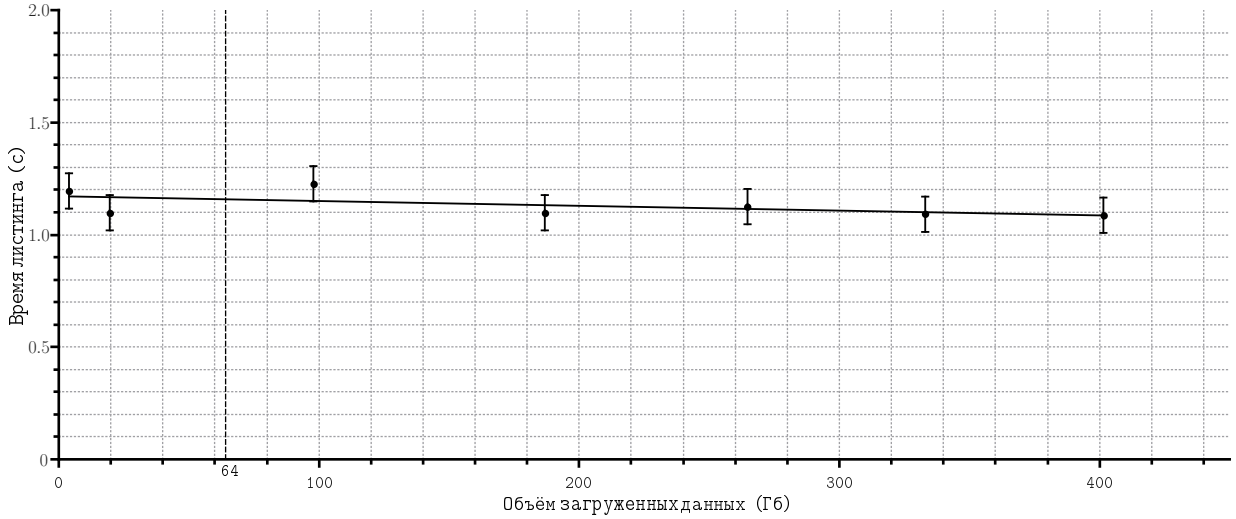

Рис. 14: Зависимость среднего времени листинга директории от объёма данных для модели документов-версий

Таблица 11: Средние времена поиска файла по краткому имени для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	13,23	14,90	14,75	14,37	14,54	14,67	14,40

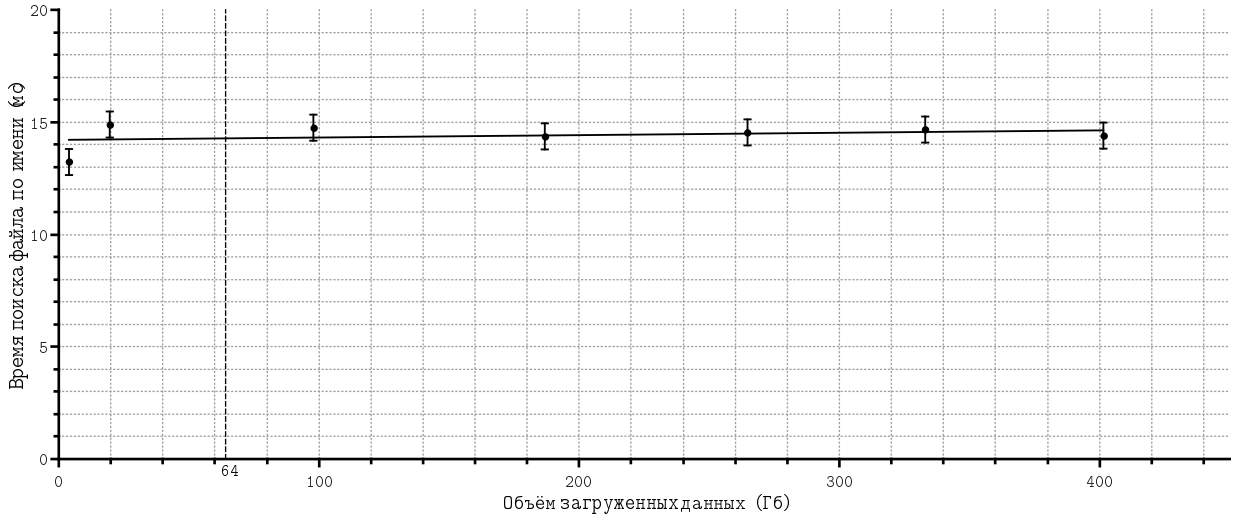

Рис. 15: Зависимость среднего времени поиска файла по краткому имени от объёма данных для модели документов-версий

Таблица 12: Средние времена поиска файла по краткому имени в актуальном состоянии данных для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	14,25	12,53	12,78	11,92	12,20	13,50	12,60

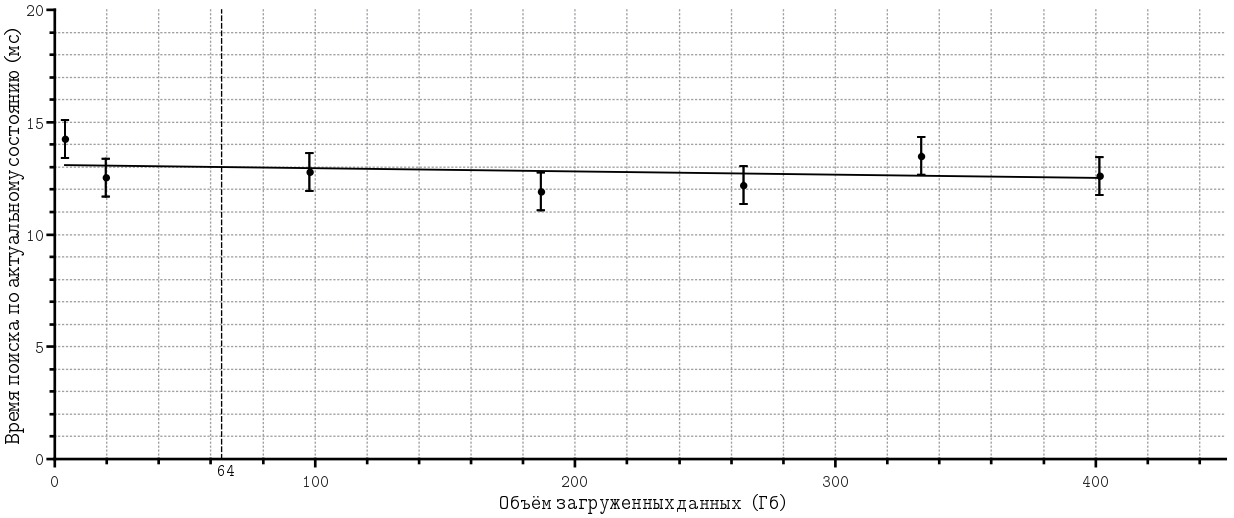

Рис. 16: Зависимость среднего времени поиска файла по краткому имени в актуальном состоянии данных от объёма данных для модели документов-версий

Таблица 13: Средние времена поиска файла по краткому имени в заданном срезе для модели документов-версий

Объём в гигабайтах	3,921	19,583	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	12,50	12,42	12,39	12,37	12,40	12,44	12,37

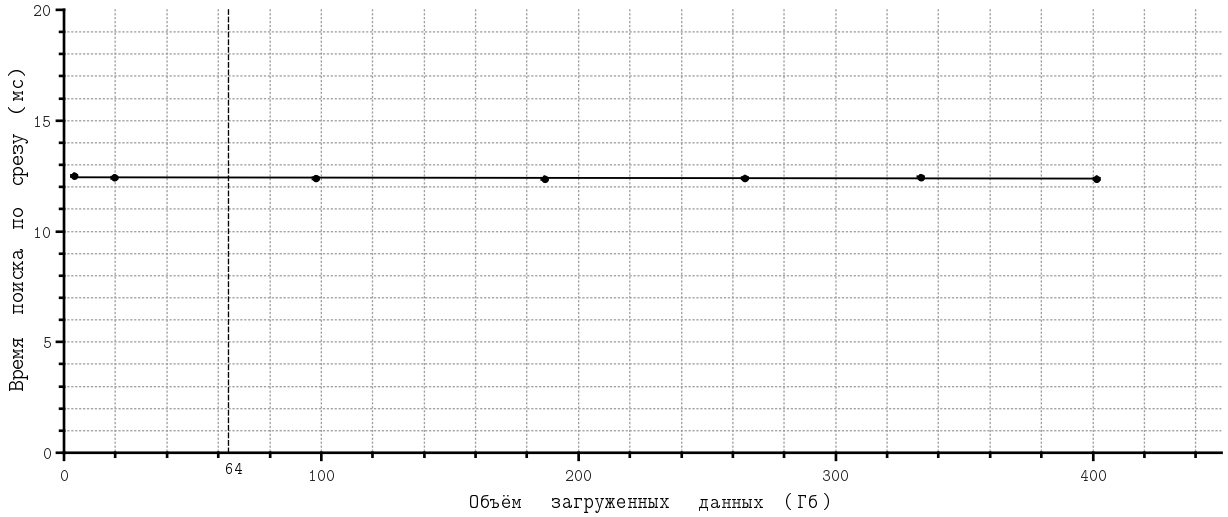

Рис. 17: Зависимость среднего времени поиска файла по краткому имени в заданном срезе от объёма данных для модели документов-версий

7.3 Времена исполнения запросов для модели с массивом номеров срезов

Таблица 14: Времена загрузки для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39,162	97,842
Время в секундах (1 срез)	1 159	4 775	11 401	19 601
Время в секундах (2 срез)	395	1 620	3 881	6 667
Объём в гигабайтах	186,788	264,697	332,993	401,462
Время в секундах (1 срез)	52 587	71 285	89 818	108 548
Время в секундах (2 срез)	17 880	24 233	30 524	36 926

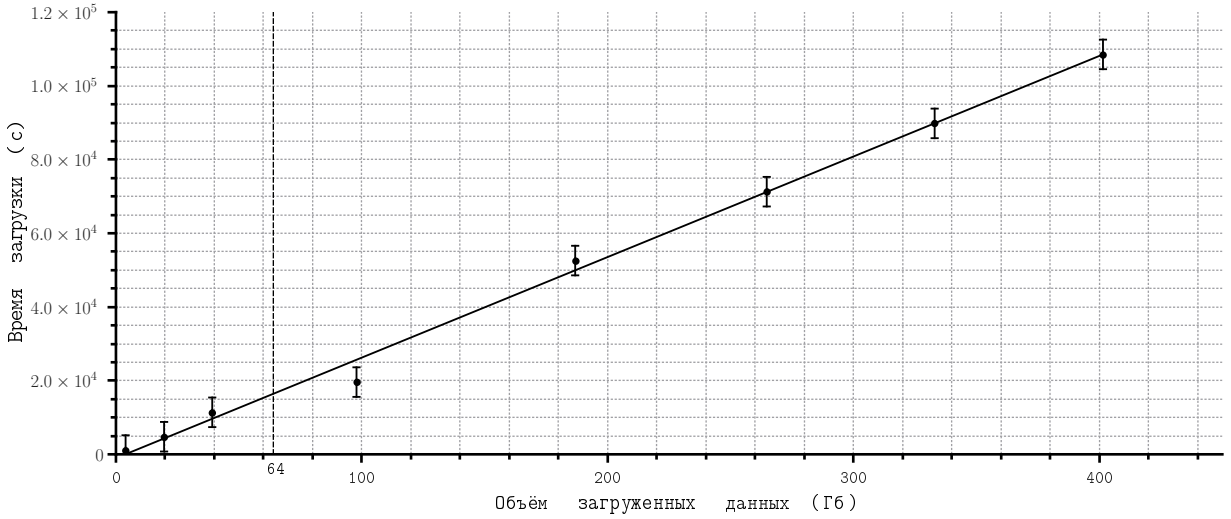

Рис. 18: Зависимость времени загрузки от объёма данных для модели с массивом номеров срезов

В отличие от модели документов-версий доля времени на переиндексацию константна и относительно мала.

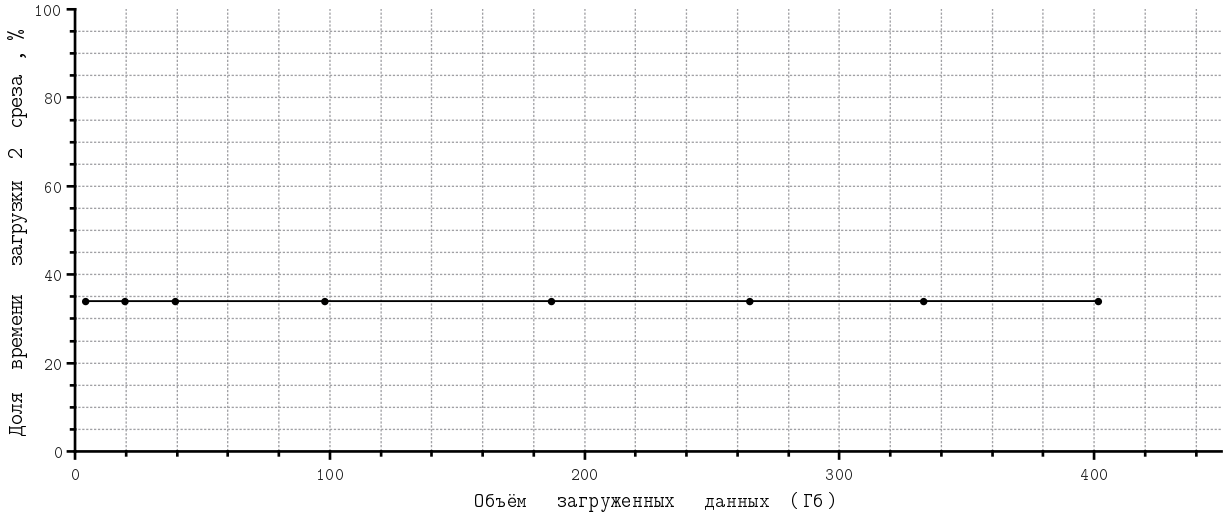

Рис. 19: Доля времени загрузки второго среза по сравнению с первым для модели с массивом номеров срезов

Таблица 15: Средние времена поиска файла по полному пути для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	12.46	11.84	13.29	19.02	12.29	12.63	12.81	16.15

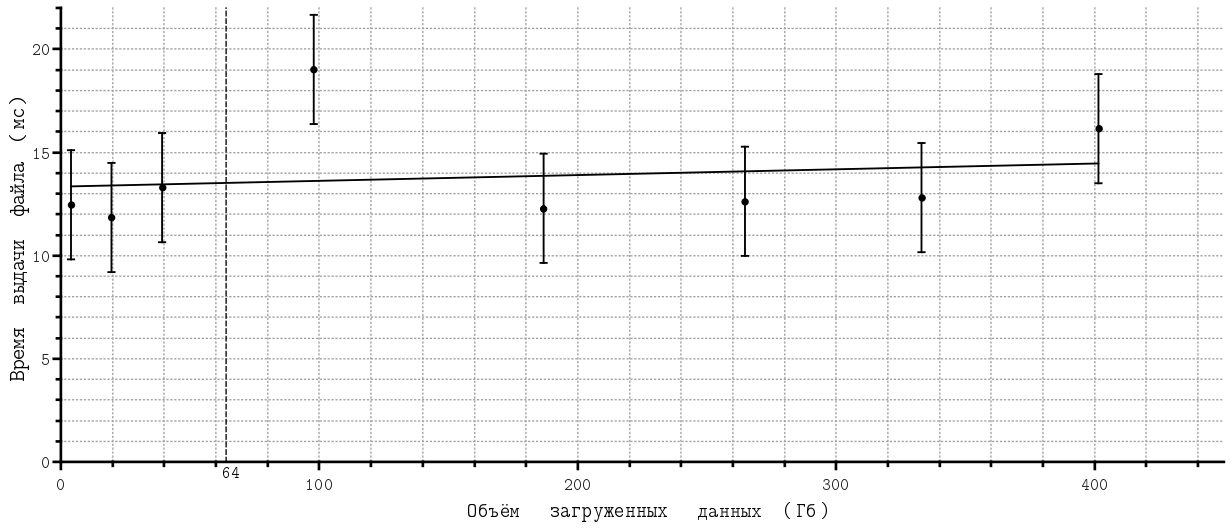

Рис. 20: Зависимость среднего времени поиска файла по полному пути от объёма данных для модели с массивом номеров срезов

Таблица 16: Средние времена полнотекстового поиска для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в секундах	1,61	1,52	1,65	1,96	1,64	1,58	1,62	1.75

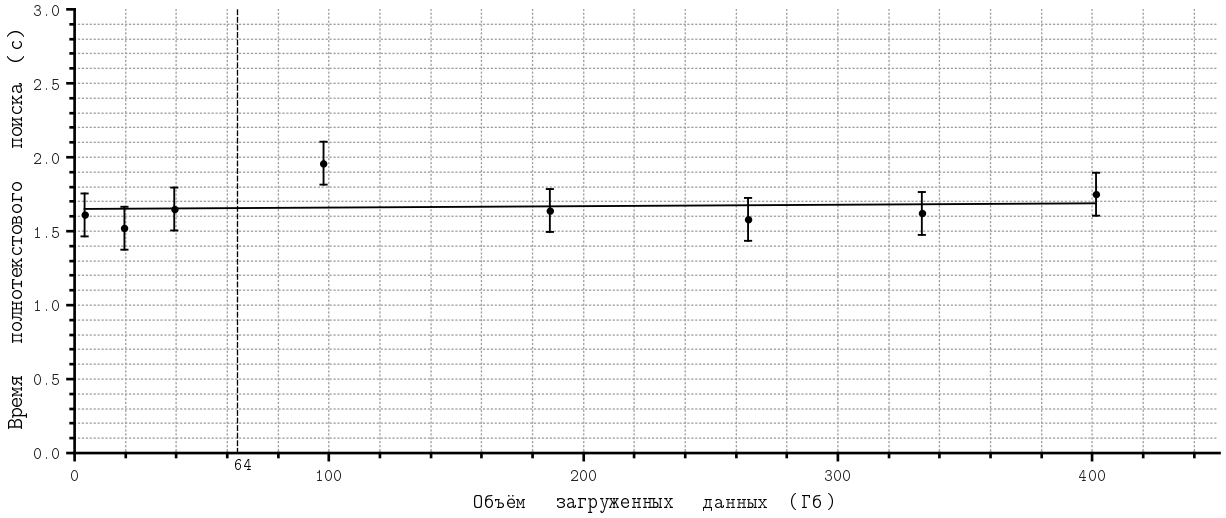

Рис. 21: Зависимость среднего времени полнотекстового поиска от объёма данных для модели с массивом номеров срезов

Таблица 17: Средние времена полнотекстового поиска с указанием директории для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39,162	97,842	186,788	264,697	332,993	401,462
Время в секундах	2,54	2,48	2,48	2,42	2,38	2,46	2,52	2,55

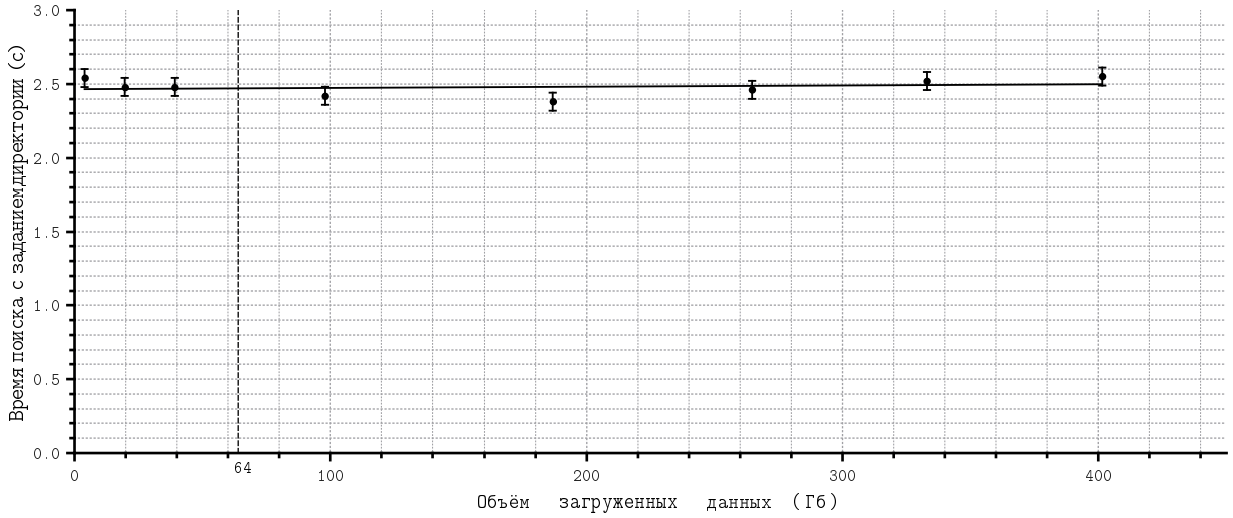

Рис. 22: Зависимость среднего времени полнотекстового поиска с с указанием директории от объёма данных для модели с массивом номеров срезов

Таблица 18: Средние времена листинга директории для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в секундах	1,12	1,08	1,02	1,14	1,12	1,10	1,17	1.04

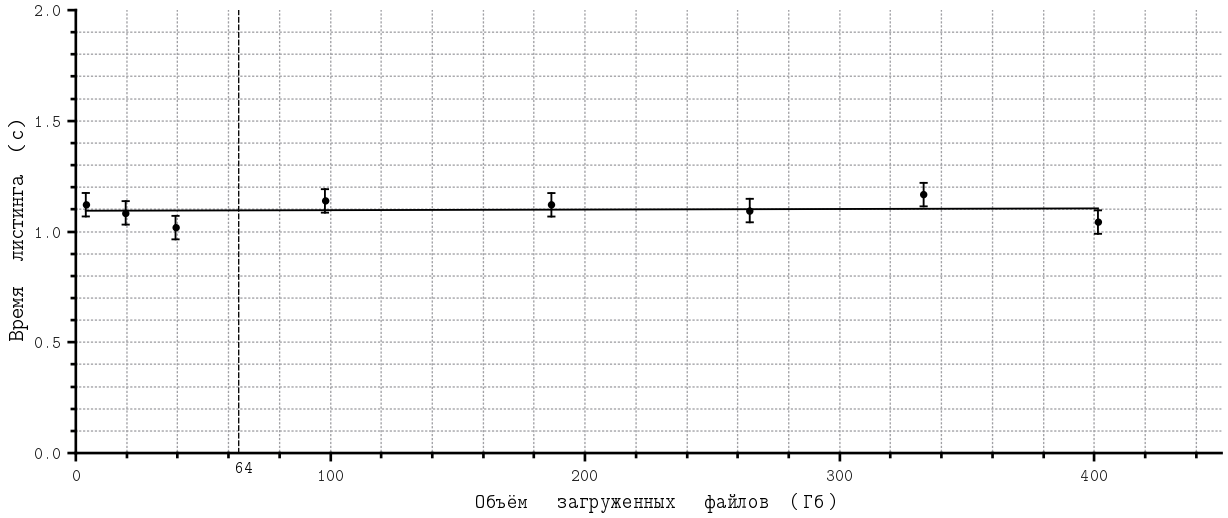

Рис. 23: Зависимость среднего времени листинга директории от объёма данных для модели с массивом номеров срезов

Таблица 19: Средние времена поиска файла по краткому имени для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	12.55	12.82	12.55	14,54	14,04	14.32	13.56	14.67

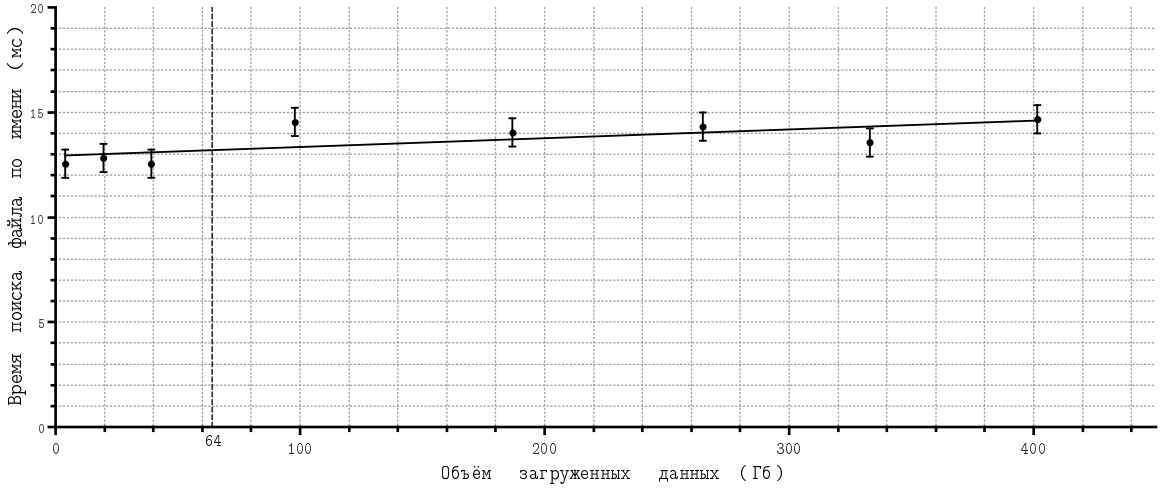

Рис. 24: Зависимость среднего времени поиска файла по краткому имени от объёма данных для модели с массивом номеров срезов

Таблица 20: Средние времена поиска файла по краткому имени в заданном срезе для модели с массивом номеров срезов

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	12,05	12,30	13,07	14,15	14,89	16,35	17,42	18,50

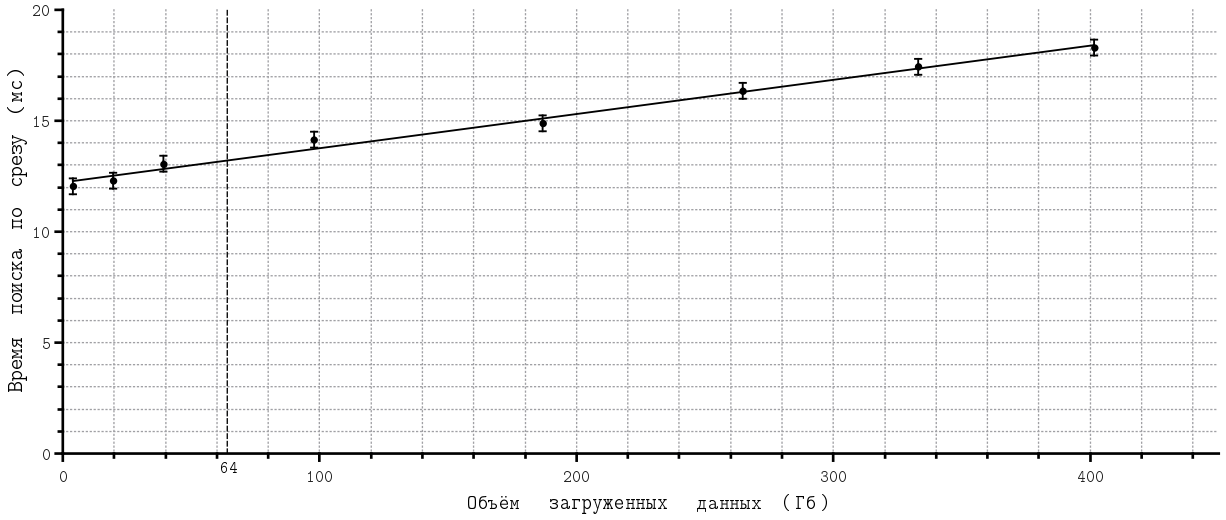

Рис. 25: Зависимость среднего времени поиска файла по краткому имени в заданном срезе от объёма данных для модели с массивом номеров срезов

Как видно, здесь время имеет тенденцию к линейному росту по объёму данных и количеству срезов. Ср. с рис. 17. Тем не менее, рост очень слабый и при рассмотренных объёмах данных им можно пренебречь.

7.4 Времена исполнения запросов для модели родитель-ребёнок

Таблица 21: Времена загрузки для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39,162	97,842
Время в секундах (1 срез)	1134	4699	10 183	19 289
Время в секундах (2 срез)	253	938	2 128	4 131
Объём в гигабайтах	186,788	264,697	332,993	401,462
Время в секундах (1 срез)	48 096	68 175	85 177	103 424
Время в секундах (2 срез)	9 847	14 355	19 128	21 174

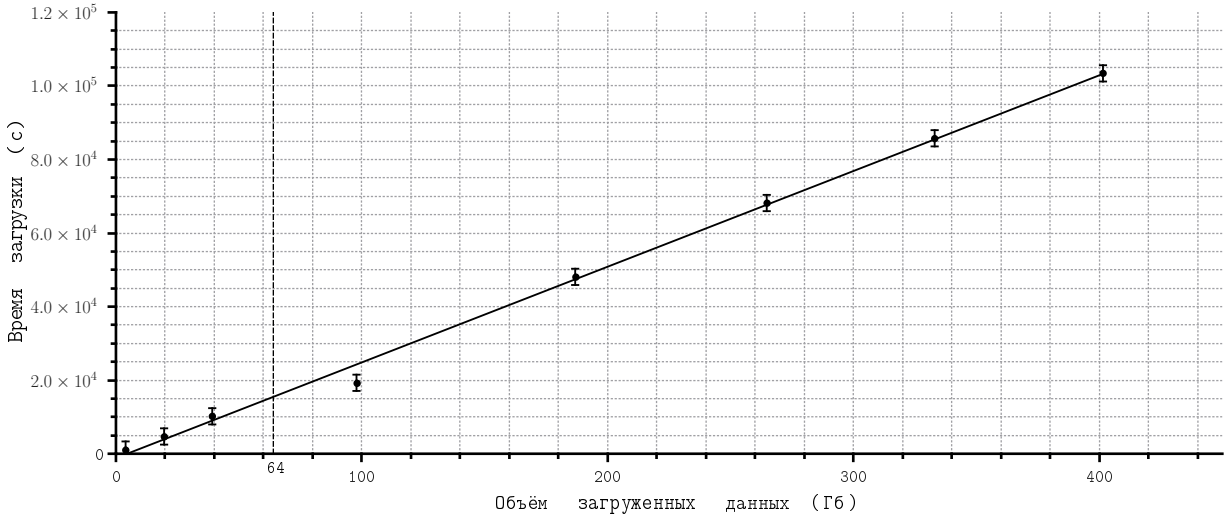

Рис. 26: Зависимость времени загрузки от объёма данных для модели родитель-ребёнок

В отличие от модели документов-версий доля времени на переиндексацию константна и относительно мала.

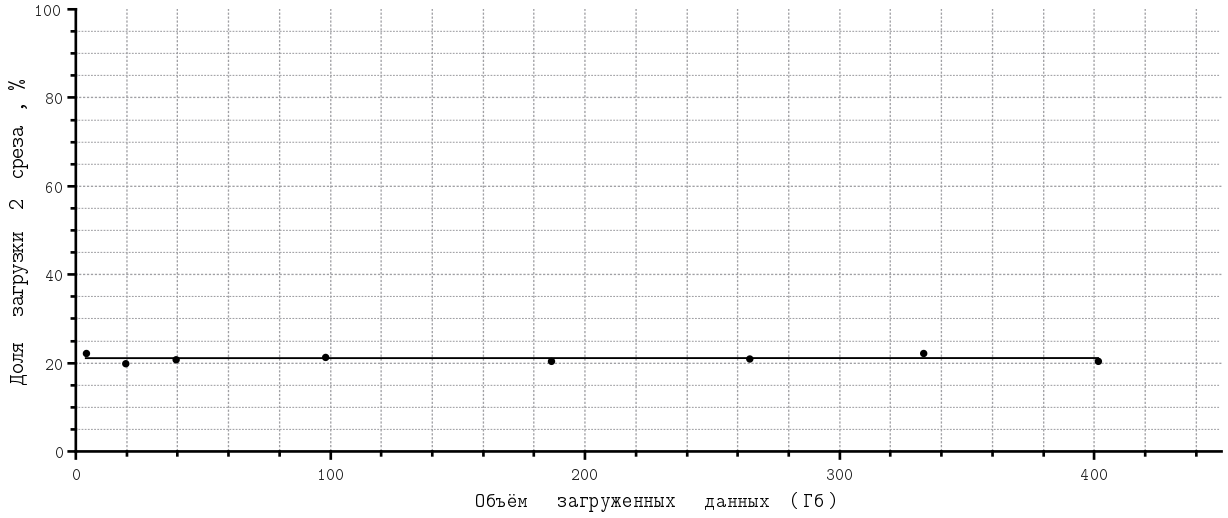

Рис. 27: Доля времени загрузки второго среза по сравнению с первым для модели родитель-ребёнок

Таблица 22: Средние времена поиска файла по полному пути для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	12.35	12.57	11.38	12.13	11.79	12.25	12.62	11.82

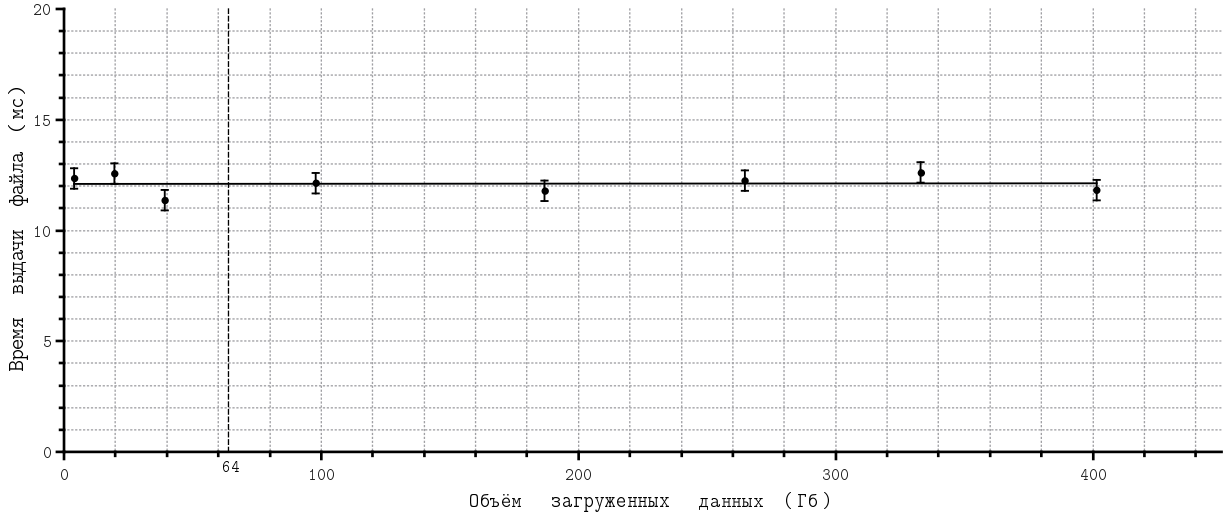

Рис. 28: Зависимость среднего времени поиска файла по полному пути от объёма данных для модели родитель-ребёнок

Таблица 23: Средние времена полнотекстового поиска для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в секундах	1,52	1,49	1,46	1,68	1,57	1,46	1,50	1.58

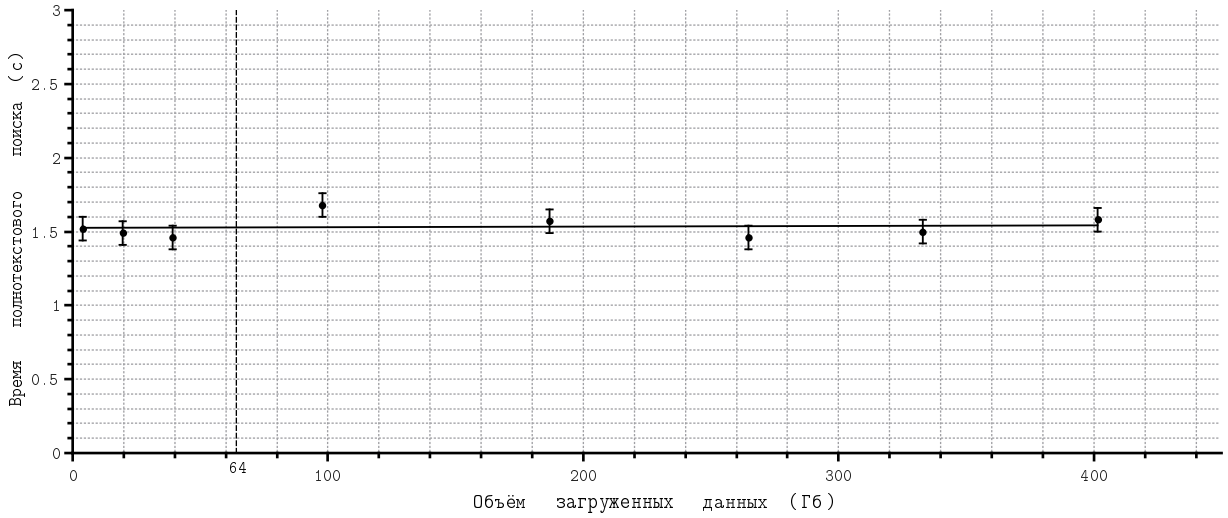

Рис. 29: Зависимость среднего времени полнотекстового поиска от объёма данных для модели родитель-ребёнок

Таблица 24: Средние времена полнотекстового поиска с указанием директории для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39,162	97,842	186,788	264,697	332,993	401,462
Время в секундах	2,42	2,51	2,25	2,47	2,41	2,35	2,45	2,27

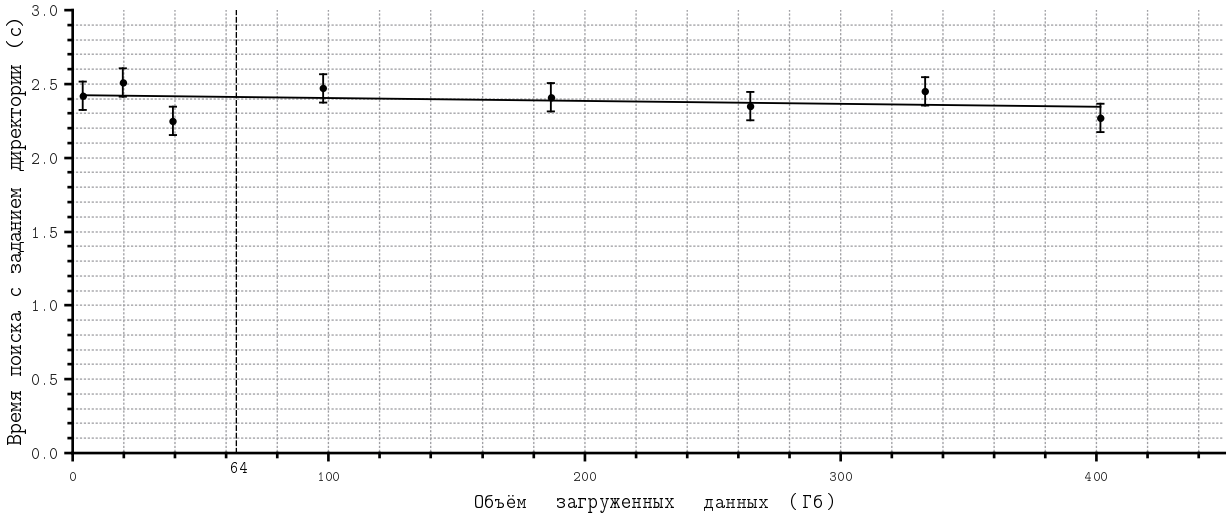

Рис. 30: Зависимость среднего времени полнотекстового поиска с с указанием директории от объёма данных для модели с массивом номеров срезов

Таблица 25: Средние времена поиска файла по краткому имени для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	13.40	13.83	11.98	14,25	13,53	13,62	12,34	14.10

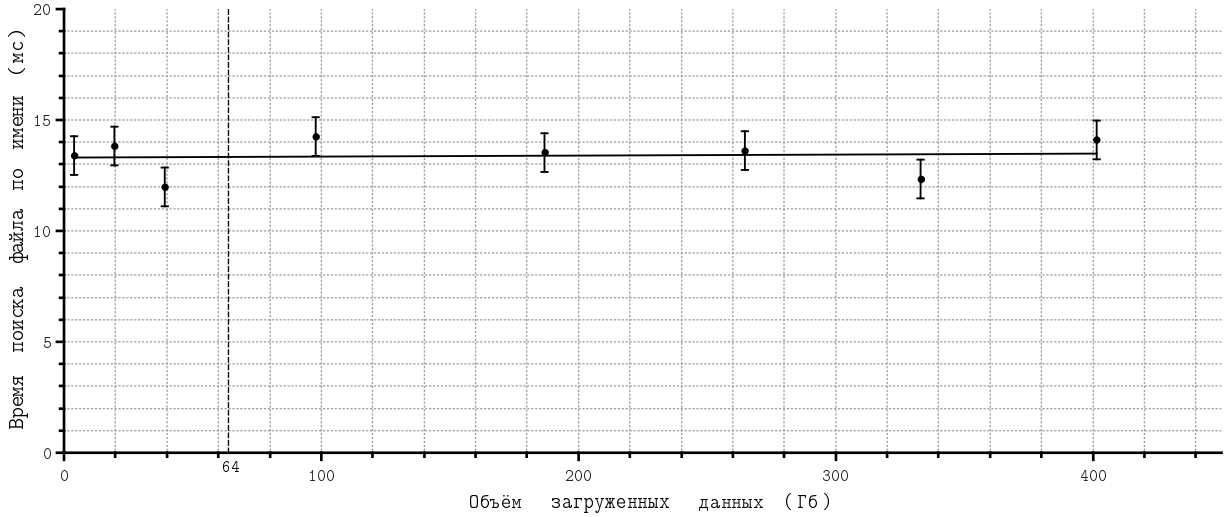

Рис. 31: Зависимость среднего времени поиска файла по краткому имени от объёма данных для модели родитель-ребёнок

Таблица 26: Средние времена поиска файла по краткому имени в заданном срезе для модели родитель-ребёнок

Объём в гигабайтах	3,921	19,583	39.162	97,842	186,788	264,697	332,993	401,462
Время в миллисекундах	16.29	18.06	16.62	17.56	16.44	18.61	17.2	16.77

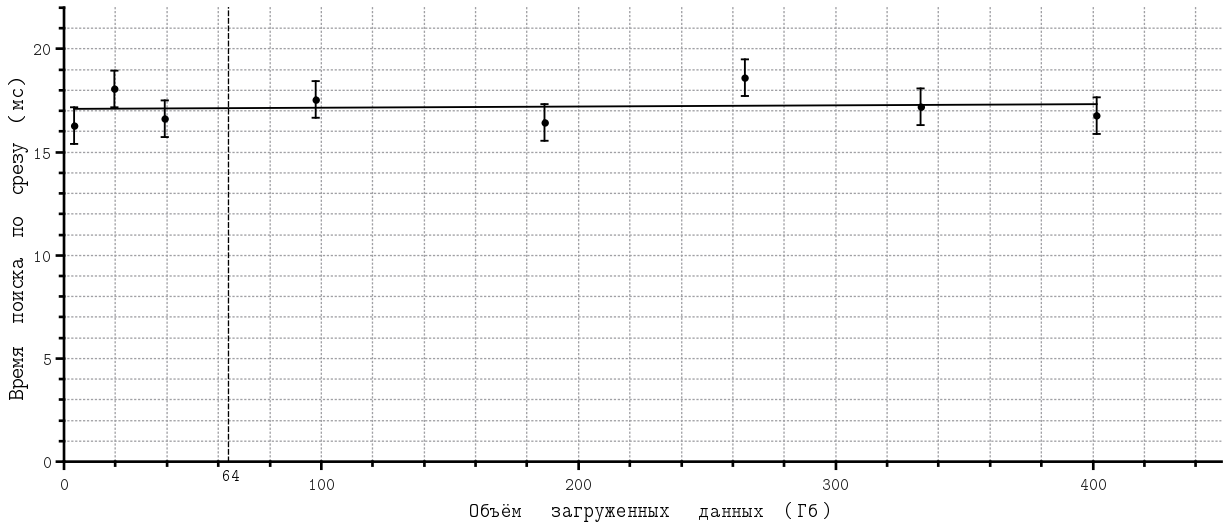

Рис. 32: Зависимость среднего времени поиска файла по краткому имени в заданном срезе от объёма данных для модели родитель-ребёнок

7.5 Размер индекса

Таблица 27: Размер хранилища Elasticsearch для двух срезов на диске, Гб

Объём индексируемых файлов	3,921	19,583	39,162	97,842	186,788	264,697	332,993	401,462
Индекс, документы-версии	11,096	66,848	—	254,288	484,770	688,212	799,183	1007,670
Индекс, «номера срезов»	5,55	26,63	62,91	173,97	248,53	381,164	499,49	586,135
Индекс, «родитель-ребёнок»	5,670	33,560	72,028	128,255	294,56	417,427	532,79	618,25

7.6 Сравнение моделей

Исходя из того, что время поиска для всех трёх моделей приближённо константное, имеет смысл сравнить средние значения для имеющейся выборки.

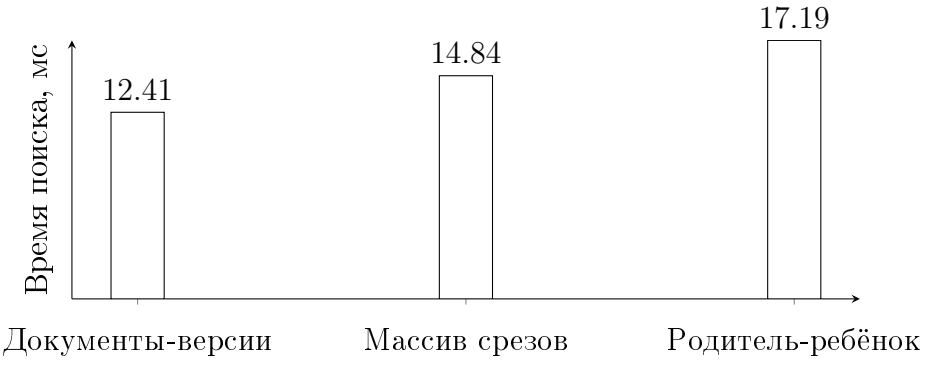

Рис. 33: Сравнительное среднее время поиска в заданном срезе

Видно, что быстрее всего поиск у модели документов-версий, медленнее всего — у модели родитель-ребёнок (как и предсказывалось теоретически), среднее время поиска в 1.4 раза больше, чем у модели документов-версий. Несмотря даже на наличие слабого тренда на линейный рост по количеству срезов, модель с массивом срезов всё же выгоднее модели родитель-ребёнок.

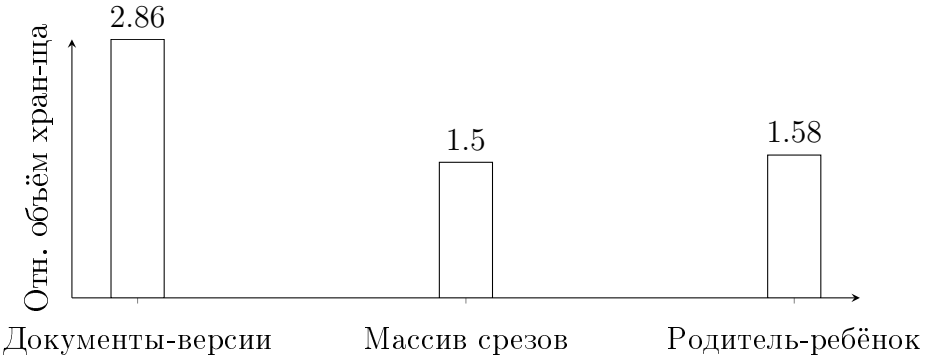

Рис. 34: Сравнительное среднее отношение размера хранилища Elasticsearch к исходным данным для двух срезов

Видно, что модель массива срезов наиболее экономна в отношении диска. Большой размер индекса для документов-версий ожидаем ввиду отсутствия дедупликации.

Следующий график приведён для объёмов данных > 100 Гб

Рис. 35: Сравнительное среднее отношение времени загрузки второго идентичного среза по отношению к первому

8 Выводы

Подход к моделированию иерархии путём использования иерархического токенизатора для пути оправдал теоретические ожидания. Время всех основных операций, как связанных с иерархией напрямую, так и не связанных, практически не зависит от объёма данных в индексе. Изменения в индексе, не касающиеся обработки и хранения пути, не оказывают существенного влияния на результаты.

Поиск с указанием директории оказался стабильно в 1.5 – 2 раза медленнее простого полнотекстового поиска.

Сравнительный анализ трёх моделей версионирования показал, что все они имеют практически константное время поиска по состоянию системы. Время первичной загрузки файла не отличается существенно между моделями. Лидером по среднему времени полнотекстового поиска по срезу оказалась модель документов-версий, лидером по экономии дискового пространства и памяти — модель с массивом номеров срезов, а лидером по времени вторичной индексации — модель родителя-ребёнка. Эти результаты согласуются с выводами, полученными теоретически.

Необходимо иметь в виду, что экономия дискового пространства, получаемая при использовании моделей с отношением «один ко многим» (модель родителя-ребёнка и модель с массивом номеров срезов), достигается за счёт не изменившихся между срезами файлов.

Таким образом, модель документов-версий можно рекомендовать для данных, меняющихся относительно быстро по сравнению с частотой создания срезов, а модели, использующие отношение «один ко многим» — для данных, меняющихся медленно. Заметим, что преимущества быстрой переиндексации особенно важны при удалении срезов.

Заключение

В работе проанализированы подходы к моделированию иерархии в документоориентированной системе, предложен подход к моделированию иерархии и три подхода к моделированию временных состояний (версий). Проведено исследование этих подходов с точки зрения производительности, причём особое внимание уделялось масштабируемости, то есть были выявлены зависимости скорости выполнения запросов от объёма обрабатываемых данных.

Реализация программы, осуществляющей тестовые запросы, продемонстрировала хорошую производительность и масштабируемость подхода к моделированию иерархии и, следовательно, его пригодность к использованию в документоориентированных системах. Подходы к версионированию проявили себя по-разному в зависимости от характера данных и вида запросов.

Работа нашла практическое применение в реализации «каталога данных» продукта Acronis Backup Advanced 12.

Возможным направлением дальнейшего развития данной темы могло бы стать исследование влияния сетевых задержек в распределённом кластере на время выполнения запросов для различных моделей.

Список литературы

[1] The MongoDB 3.4 Manual. [Электронный ресурс]. URL: https://docs.mongodb.com/manual – 2017

[2] *Ma K., Yang B., Abraham A.* Toward full-text searching middleware over hierarchical documents // 13th International Conference on Intelligent Systems Design and Applications (ISDA), 2013. P. 194 – 198.

[3] *Castelltort A., Laurent A.* Representing history in graph-oriented NoSQL databases: a versioning system // Eighth International Conference on Digital Information Management (ICDIM), 2013. P. 228 – 234.

[4] *Celko J.* Trees and hierarchies in SQL for smarties. Burlington: Morgan Kaufmann, 2012. 238 p.

[5] *Tropashko V.* SQL design patterns: expert guide to SQL programming. Kittrell: Rampant Techpress, 2007. Vol. 4. 254 p.

[6] *Gormley C., Tong Z.* Elasticsearch: the definitive guide. Sebastopol: O'Reily Media, 2015. 719 p.

[7] Stackexchange performance. [Электронный ресурс]. URL: https://stackexchange.com/performance – 2017

[8] DB-Engines ranking of search engines. [Электронный ресурс]. URL: https://db-engines.com/en/ranking/search+engine – 2017

[9] Руководство пользователя Acronis Backup Advanced 11.7. [Электронный ресурс]. URL: https://www.acronis.com/ru-ru/download/docs/aba11.7/userguide/ – 2017

Приложение А. Описания индексов в формате JSON

А.1 Настройки анализа

```
{
    "settings": {
        "analysis": {
            "analyzer": {
                "path-analyzer": {
                    "type": "custom",
                    "tokenizer": "path-tokenizer"
                },
                "path-reverse-analyzer" : {
                    "type": "custom",
                    "tokenizer": "path-reverse-tokenizer"
                }
            },
            "tokenizer": {
                "path-tokenizer": {
                    "type": "path_hierarchy",
                    "delimiter": "/",
                    "reverse": false
                },
                "path-reverse-tokenizer": {
                    "type": "path_hierarchy",
                    "delimiter": "/",
                    "reverse": true
                }
            }
        }
    }
}
```

A.2 Файловый индекс для модели документов-версий

```
{
    "mappings": {
        "file": {
            "properties": {
                "actual": {
                    "type": "bool",
                    "index": "not_analyzed"
                },
                "slice": {
                    "type": "long",
                    "index": "not_analyzed"
                },
                "machine": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "path" : {
                    "type": "string",
                    "fields": {
                        "searchable": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "searchable_reverse": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-reverse-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "listable" : {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                },
                "mtime": {
                    "type": "date",
                    "format": "yyyy-MM-dd HH:mm:ss.SSSSSSSSS ZZZZZ",
                    "index": "not_analyzed"
                },
                "body" : {
                    "type": "string",
                    "index": "analyzed"
                }
            }
        }
    }
}
```

A.3 Файловый индекс для модели родитель-ребёнок

```
{
    "mappings": {
        "version": {
            "_parent": {
                "type": "file"
            },
            "properties": {
                "slice": {
                    "type": "long",
                    "index": "not_analyzed"
                }
            }
        },
        "file": {
            "properties": {
                "machine": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "path" : {
                    "type": "string",
                    "fields": {
                        "searchable": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "searchable_reverse": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-reverse-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "listable" : {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                },
                "mtime": {
                    "type": "date",
                    "format": "yyyy-MM-dd HH:mm:ss.SSSSSSSSS ZZZZZ",
                    "index": "not_analyzed"
                },
                "body" : {
                    "type": "string",
                    "index": "analyzed"
                }
            }
        }
    }
}
```

A.4 Директорный индекс для модели документов-версий

```
{
    "mappings": {
        "dir": {
            "properties": {
                "actual": {
                    "type": "bool",
                    "index": "not_analyzed"
                },
                "slice": {
                    "type": "long",
                    "index": "not_analyzed"
                },
                "machine": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "path" : {
                    "type": "string",
                    "fields": {
                        "searchable": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "listable" : {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                }
            }
        }
    }
}
```

A.5 Директорный индекс для моделей с отношением «один ко многим»

```
{
    "mappings": {
        "dir": {
            "properties": {
                "slice": {
                    "type": "long",
                    "index": "not_analyzed"
                },
                "machine": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "path" : {
                    "type": "string",
                    "fields": {
                        "searchable": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "listable" : {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                }
            }
        }
    }
}
```

A.6 Файловый индекс для модели с номерами срезов

```
{
    "mappings": {
        "file": {
            "properties": {
                "slice": {
                    "type": "long",
                    "index": "not_analyzed"
                },
                "machine": {
                    "type": "string",
                    "index": "not_analyzed"
                },
                "path" : {
                    "type": "string",
                    "fields": {
                        "searchable": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "searchable_reverse": {
                            "type": "string",
                            "index": "analyzed",
                            "analyzer": "path-reverse-analyzer",
                            "search_analyzer": "keyword"
                        },
                        "listable" : {
                            "type": "string",
                            "index": "not_analyzed"
                        }
                    }
                },
                "mtime": {
                    "type": "date",
                    "format": "yyyy-MM-dd HH:mm:ss.SSSSSSSSS ZZZZZ",
                    "index": "not_analyzed"
                },
                "body" : {
                    "type": "string",
                    "index": "analyzed"
                }
            }
        }
    }
}
```

Приложение Б. Запросы к системе на языке Elasticsearch Query DSL

Б.1 Загрузка файлов

```
POST /_bulk

{"create": {"_index": "dir_index", "_type": "dir", "_id": "X1" }}
{"actual": true, "slice": 10, "machine": "m1", "path": "XXXXXX"}
{"create": {"_index": "file_index", "_type": "file", "_id": "Y1" }}
{"actual": true, "slice": 10, "machine": "m1", "path": "XXXXXX/YYYYYY", "size": 1024,
    "owner": "alexey", "group": "alexey", "mtime":"2016-11-15 16:25:15.800955432 +0300",
        "body": "Hello world!"}
```

Б.2 Полнотекстовый поиск

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "match": {"body": "XXXXXX"}
    }
}
```

Б.3 Полнотекстовый поиск с указанием директории

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "bool": {
            "filter": {
                "bool": {
                    "must": [
                        {"term": {"machine": "m1"}},
                        {"term": {"path.searchable": "XXXXXX"}}
                    ]
                }
            },
            "must": {
                "match": {"body": "XXXXXX"}
            }
        }
    }
}
```

Б.4 Листинг директории

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "constant_score": {
            "filter": {
                "bool": {
                    "must": [
                        {"term": {"machine": "m1"}},
                        {"term": {"path.listable": "XXXXXX"}}
                    ]
                }
            }
        }
    }
}
```

Б.5 Поиск файла по имени (краткому, полному)

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "constant_score": {
            "filter": {
                "term": {"path.searchable_reverse": "XXXXXX"}
            }
        }
    }
}
```

Б.6 Поиск файла по имени в заданном срезе

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "constant_score": {
            "filter": {
                "bool": {
                    "must": [
                        {"term": {"slice": 9}},
                        {"term": {"path.searchable_reverse": "XXXXXX"}}
                    ]
                }
            }
        }
    }
}
```

Б.7 Поиск файла по имени в актуальном состоянии

```
GET /file_index/_search?filter_path=hits.total,hits.hits._source

{
    "query": {
        "constant_score": {
            "filter": {
                "bool": {
                    "must": [
                        {"term": {"actual": true}},
                        {"term": {"path.searchable_reverse": "XXXXXX"}}
                    ]
                }
            }
        }
    }
}
```