Санкт-Петербургский государственный университет

Кафедра Технологии Программирования

Иванов Алексей Олегович

Анализ текстов судебных решений

Выпускная квалификационная работа бакалавра

Направление 01.03.02

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель:
старший преподаватель Севрюков С.Ю.

Санкт-Петербург
2019

Оглавление

Введение

Актуальность. За последние десять–пятнадцать лет произошел взрывной рост объема мировых данных в сети Интернет. Этому много причин, но основная заключается в том, что Интернет все больше входит в повседневную жизнь людей. Наибольший вклад в этот рост вносят данные веб-приложений. Многие вещи, которые раньше можно было сделать только офлайн, сейчас можно делать через онлайн-сервисы, например, теперь существует возможность покупать авиа и железнодорожные билеты через интернет. При этом создается большое количество данных, которое необходимо обрабатывать. В связи с увеличением количества хранимых данных, повышаются требования к скорости загрузки и обработки данных.

Одна из сфер, где также замечен рост объема данных, – правовая информатика. Правовая информатика — это прикладная юридическая наука, исследующая право и правовую систему общества с точки зрения информатики. Существует множество веб-сервисов, которые предоставляют доступ к аналитике юридических документов. Под анализом юридических документов понимается правовое исследование документа на предмет соответствия действующему законодательству, оценки юридической грамотности документа, выделения ссылок на другие юридические документы, а также выделения окраски и смысла вынесенного решения. Юридические документы являются неструктурированными данными, поэтому с точки зрения анализа текстов, ключевой задачей является извлечение знаний из огромного массива информации. Одним из способов извлечения знаний является информационный поиск. Тексты судебных решений судов Российской Федерации, а также тексты кодексов в различных изданиях со всеми правками занимают сотни ГБ памяти, а значит, для обработки поискового запроса по ним необходимо рассмотреть большое количество документов. Точность в подобных вопросах играет ключевую роль, поэтому разработка информационно-поисковой системы, которая будет удовлетворять запросам пользователя, является актуальной задачей на стыке правовой информатики и

информационного поиска.

1. Постановка задачи

В 2007 году вышла научная статья под названием «Network Analysis and the Law»[6]. В статье описывалось построение графа из 26681 решений Верховного суда с 1791 по 2005 год. При помощи этого графа было проведено исследование, результатом которого стала методология, которая способна помочь ученым измерить, насколько юридически важным является решение Верховного суда. Понимание важности решения необходимо, чтобы дать оценку правомерности вынесенного решения Верховного суда. Ученые часто отмечают, что необходимо оценивать юридические дела в контексте смежных дел. Использование цитирования и отсылок к другим юридическим документам определяет контекст и позволяет выявлять юридическую важность вынесенного решения.

В контексте вышеуказанной статьи в 2018 году деканом юридического факультета СПбГУ была предложена задача реализации программного решения. Оно должно было бы совмещать в себе функционал анализа текстов судебных актов на предмет семантических ссылок и выполнения аналитических запросов.

Задача поиска подходящего инструмента для эффективной индексации и поиска данных является важной частью разработки этого программного продукта. Предметом исследования является применимость и масштабируемость платформ полнотекстового поиска в веб-сервисе, который является частью проекта "Юридическая аналитика". Проект "Юридическая аналитика", разрабатываемого в рамках учебной практики СПбГУ, предоставляет услуги по расширенной аналитике судебных актов, что позволяет существенно сократить время, затрачиваемое прикладными специалистами для анализа судебных актов. В данной работе рассмотрены популярные библиотеки информационно-поисковых систем, их архитектурные подходы, описаны основные характеристики и особенности, а также рассмотрены преимущества и недостатки каждого из них. На основе этого анализа было принято решение, какое программное обеспечение для поиска информации стоит использовать и как можно улучшить качество поиска.

Целью данной работы является усовершенствование архитектуры поискового сервиса проекта ”Юридическая аналитика”, исходя из этого были определены следующие задачи:

1. Изучение существующих информационно-поисковых систем и их возможностей для полнотекстового поиска.

2. Изучение специфики юридических документов и особенностей их поиска.

3. Выбор программного обеспечения, подходящего под требования и ограничения, накладываемые на веб-сервис, описанные в техническом задании к проекту проекту «Юридическая аналитика».

4. Разработка архитектуры поискового веб-сервиса.

1.1. Описание планируемой архитектуры

Продукт «Юридическая аналитика» состоит из двух частей: Первая часть — программная библиотека с алгоритмами для анализа текста. Вторая часть — веб-сервис, который предоставляет интерфейс к функционалу библиотеки.

Схема планируемой архитектуры отражена на Рис. 1

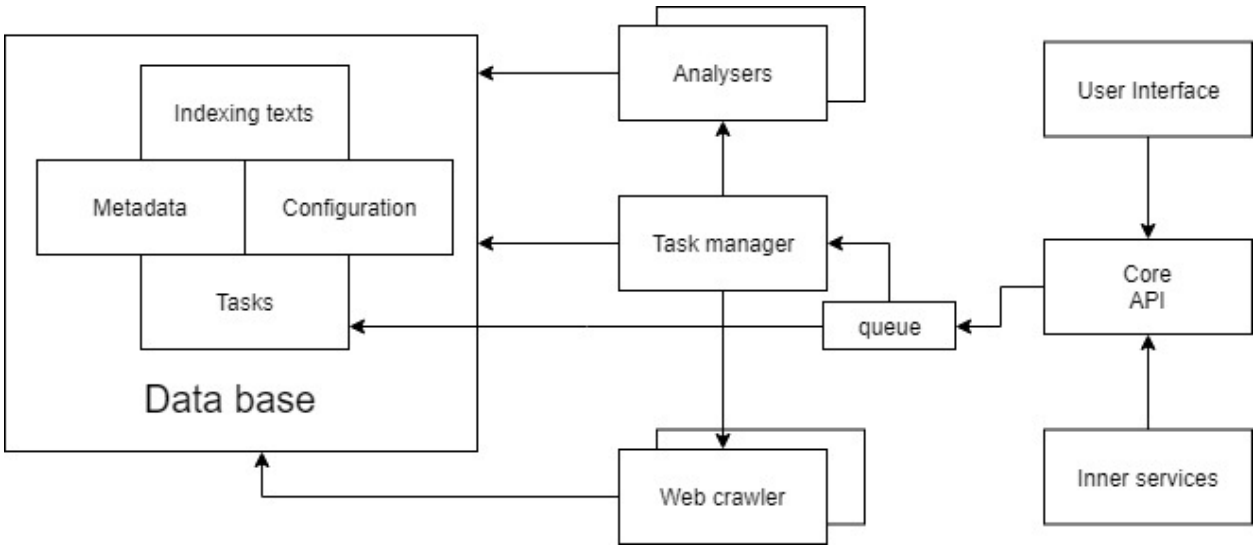

Рис. 1: Схема

1. UI — веб-интерфейс
2. Inner services — внутренние сервисы
3. Core API — публичный API для взаимодействия с пользователем
4. Queue — очередь для хранения задач
5. Task manager — планировщик задач
6. Analyzers — алгоритмы для анализа текстов
7. Web crawler — поисковый робот, сканирующий доступные юридические документы
8. Database — сервис хранения информации, состоящий из:
   (a) Tasks — история задач поступающая от пользователей
   (b) Configuration — текущая конфигурация системы
   (c) Metadata — метаданные текстов (особенности)
   (d) Indexing texts — проиндексированные тексты и их индексы

Компонента хранения и поиска информации занимает важное место в планируемой архитектуре и от нее напрямую зависит производительность системы и удовлетворенность пользователей, исходя из этого к функциональности данной компоненты выдвигаются следующие требования:

1. Изменение возможностей поиска через API поискового сервиса
2. Репликация и шардирование необходимы для масштабирования информационно-поискового сервиса
3. Возможность полнотекстового поиска:
   (a) Гибкая система анализа слов – нормализация, стемминг, деление на N-граммы, проверка орфографии, поддержка стоп-слов и синонимов

(b) Разные виды поиска необходимы для улучшения метрик полноты и точности при использовании поэтапного поиска для расширения пользовательского запроса:

   i. Поиск по многим полям документа

   ii. Префиксный поиск

   iii. Нечеткий поиск

(c) Сортировка результатов поиска

(d) Возможность получения фасетов – понадобится для использования на слое UI, чтобы пользователь мог выбирать нужную ему категорию или фильтровать нужные ему документы на сайте без редактирования поисковой строки. Фасеты можно считать по временным интервалам или по кодексам, на которых найдены решения Верховного суда

(e) Возможность «подсветки» фраз в найденных документах – «подсветка», которая тоже будет использоваться также на слое UI, позволяет пользователю находить слова из поисковой фразы в найденном документе

1.2. Предлагаемое решение по улучшению качества поиска

Результат работы поисковой системы может оцениваться по-разному. Исторически основными метриками оценки качества информационного поиска являются полнота и точность. Полнота – отношение числа найденных релевантных документов, к общему числу релевантных документов в базе. Точность определяется как отношение числа найденных релевантных документов, к общему числу найденных документов

Для того чтобы добиться баланса между этими метриками необходимо разработать дополнительную функциональность вместо обычного полнотекстового поиска – проверка орфографии, применение концептного поиска и поэтапного поиска.

Концептный поиск является одним из методов интеллектуального анализа текста, который используется для извлечения слов, которые могут быть важны с точки зрения описания содержания документа. Это могут быть как ссылки на другие юридические документы, название суда, географических мест, а также другие термины предметной области. Задача концептного поиска состоит в том, чтобы наиболее точно выделять по заданному слову категорию, к которой оно относится, и как следствие поле, в котором можно это слово найти. Данный подход, повышающий метрику точности поискового запроса, был впервые описан в докладе «Concept Search for eCommerce with Solr» Михаила Хлуднева на конференции «Lucene/Solr Revolution 2013» и за последнее время получил широкое распространение среди крупных E-Commerce платформ. Концептный поиск можно реализовать следующим образом: после получения искомой фразы, необходимо разделить фразу на слова, затем для каждого слова найти список полей документа, в которых каждое слово может находиться. Затем, после проведенной операции необходимо изменить пользовательский запрос при помощи наложения фильтров на найденные слова, чтобы добиться большей точности поиска. Формировать список концептов можно как вручную, так и при помощи автоматического выделения метаинформации из индексируемых документов, например, из каждого юридического документа можно извлечь, название кодекса, к которому принадлежит документ, а также дату вынесения решения.

Даже с учетом проверки орфографии и других средств анализа текста может возникнуть ситуация, когда пользовательский запрос настолько сложный, что нет таких документов, которые бы удовлетворяли бы всем ограничениям, введенным пользователем. В этом случае применим поэтапный поиск, который используется для улучшения метрики полноты. Основной задачей поэтапного поиска является вернуть пользователю хотя бы один или несколько документов (в зависимости от ограничений на минимальное количество возвращаемых документов поискового сервиса). Чтобы достигнуть этой цели необходимо разделить поиск на несколько этапов:

1. Обычный поиск

2. Поиск с проверкой орфографии

3. Поиск с исключением каких-то слов

2. Обзор

Для выбора наилучшего инструмента поиска информации необходимо провести качественный анализ библиотек и СУБД, имеющих функцию полнотекстового поиска. С этой целью были выбраны самые популярные open source поисковые платформы по версии db-engines.com[4] – Elasticsearch, Solr, Sphinx, Sonic, а также реляционная СУБД PostgreSQL, которая сейчас используется в архитектуре проекта.

2.1. Сравнение библиотек полнотекстового поиска

2.2. Sphinx search engine

Sphinx [10] – поисковый сервер, разработанный Андреем Аксёновым. Имеет весьма быструю скорость индексации и поиска, поддерживает стемминг и лемматизацию английского, русского, немецкого и других популярных языков, также допускает поиск по нескольким полям (однако количество полей ограничено 256). Следует отметить, что исходное содержимое полей не хранится в индексе Sphinx. Текст, который отправляется в Sphinx, обрабатывается, и строится инвертированный индекс, однако исходное текстовое содержание простоотбрасывается, потому что предполагается, что содержимое документов хранится где-то еще.

Данная система полнотекстового поиска поддерживает ограниченную токенизацию текста: управлять разделением текста на токены можно только при помощи списка допустимых символов.

Отличительной особенностью поискового сервиса Sphinx является нативная поддержка работы с популярными на сегодняшний день SQL базами данных – MySQL и PostgreSQL, что позволяет его использовать в обычном для веб-разработки окружении.

- **Платформа:** C++
- **Индекс:** монолитный + дельта-индекс

- **Размер индекса и скорость индексации:** индекс занимает 30% от объема исходных данных, скорость индексации – около 15 MB/c
- **Типы документов:** только текст или XML-формат
- **API:** (нативная поддержка PostgreSQL и MySQL), встроенные API для PHP, Python, Ruby, C, Java

2.3. Solr

Solr [8] – платформа полнотекстового поиска с открытым исходным кодом, основанная на проекте Apache Lucene и значительно расширяющее её возможности. Solr является самостоятельным сервером, имеющий широкие возможности. Solr поддерживает передачу документов посредством HTTP запросов в формате XML и JSON. Полностью поддерживается кластеризация и репликация на несколько серверов, расширена поддержка дополнительных полей в документах (для них поддерживаются различные стандартные типы данных), поддержка фасетного поиска и фильтрации, развитые средства конфигурирования и администрирования, а также возможности бекапа индекса в процессе работы[13].

Согласно рейтингу поисковых сервисов DB-Engines Solr широко применяется в E-commerce проектах, и за многие годы существования сформировалось крупное сообщество разработчиков, опыт которых позволяет быстро решить типовые вопросы. Solr имеет большое количество инструментов для обработки текста – 31 встроенных стеммеров, различные токенизаторы текста, более 40 фильтров для анализа текста, поддержку аббревиатур, стоп-слов, синонимов. Также нельзя не отметить важную особенность, которая важна в «near real-time» системах – атомарные обновления одного поля или нескольких полей документа без переиндексации.

Важно отметить, что Solr имеет последовательную и полную документацию, которая позволяет глубоко погрузиться в процессы, проис-

ходящие во время индексирования и поиска, без вникания в код.

- **Платформа:** Java
- **Индекс:** инкрементный индекс, но требующий операции слияния сегментов (можно параллельно с поиском)
- **Размер индекса и скорость индексации:** индекс занимает 20% от объема исходных данных, скорость индексации – около 50 Мб/сек
- **Типы документов:** текст
- **API:** Java API

2.4. Elasticsearch

Elasticsearch [5] — программная поисковая система, которая имеет множество вспомогательных инструментов для работы с данными. Построен на основе Apache Lucene так же как и Solr. Однако если Solr используется и предназначен в основном для текстового поиска, то Elasticsearch используется и в других направлениях, где он выходит за рамки задач поиска для решения лог-аналитики и визуализации текстовых данных.[2] Стек дополнительных инструментов Elasticsearch включает в себя инструменты для захвата данных (Beats), обработки/преобразования данных (Logstash), визуализации данных (Kibana) и многое другое. Другими словами, Elasticsearch может обеспечить функциональность, которая обычно зарезервирована для аналогичных платных инструментов бизнес-аналитики.

Сравнить по производительности с Solr довольно тяжело. Они базируются на одной и той же библиотеке – Lucene, и результаты тестирования очень сильно зависят от параметров запуска, индексируемых данных, а также от самих запросов.

В высоконагруженных приложениях, в которых обновление документов происходит часто очень важно иметь возможность быстро менять поле документа в индексе. Elasticsearch как и Solr поддерживают

несколько модификаторов, которые атомарно обновляют значения поля документа.[11] Это позволяет обновлять только определенные поля, что может помочь ускорить процессы индексирования в среде, где скорость добавления индекса имеет решающее значение для приложения. Одним из таких модификаторов являются – атомарные обновления. Такой подход позволяет изменять только одно или несколько полей документа без необходимости повторного индексирования всего документа. Второй подход известен как обновление на месте или «in-place update». Этот подход аналогичен атомарным обновлениям (в некотором смысле является подмножеством атомарных обновлений), но может использоваться только для обновления однозначных неиндексированных и не хранимых числовых полей на основе docValues. Одним из недостатков Elasticsearch относительно Solr как раз таки и является отсутствие обновления на месте или обновления поля без полной переиндексации документа. Также нельзя не отметить, что ElasticSearch, несмотря на открытость, по большому счету коммерческий продукт компании «Elastic», которая лицензирует куски своего кода и закрывает бесплатный доступ к очевидно необходимым модулям вроде аутентификации.

- **Платформа:** Java
- **Индекс:** инкрементный индекс, но требующий операции слияния сегментов (можно параллельно с поиском)
- **Размер индекса и скорость индексации:** индекс занимает 20% от объема исходных данных, скорость индексации – около 50 Мб/сек
- **Типы документов:** текст
- **API:** Ruby, Java, JavaScript, GO, .NET, PHP, Perl, Python

2.5. PostgreSQL full-text search

PostgreSQL [7] – свободная объектно-реляционная система управления базами данных, которая базируется на языке SQL. Есть встроенные

стеммеры, поддержка стоп-слов, синонимов, ранжирование результатов поиска В настоящее время в PostgreSQL есть только один встроенный анализатор текста, который может быть полезен для широкого круга приложений. Он распознаёт 23 типа «слов», таких как адрес электронной почты, слово из любых букв, целое со знаком и без, тег XML и т.д.

Индекс у PostgreSQL инвертированный на основе GIN (Generalized Inverted Index). Он содержат записи для всех отдельных слов (лексем) с компактным списком мест их вхождений. При поиске нескольких слов можно найти первое, а затем воспользоваться индексом и исключить строки, в которых дополнительные слова отсутствуют. Индексы GIN хранят только слова (лексемы) из значений tsvector, и теряют информацию об их весах. Таким образом для выполнения запроса с весами потребуется перепроверить строки в таблице. Индекс GIN для поиска ключей использует B-дерево. Из-за этого он не очень производителен для документов которые постоянно изменяются, так как изменения приводят к большому количеству обновлений индекса.

- **Платформа:** C
- **Индекс:** инвертированный на основе GIN (Generalized Inverted Index)
- **Размер индекса и скорость индексации:** индекс занимает 150% от объема исходных данных, скорость индексации – около 15 MB/c
- **Типы документов:** текст
- **API:** Нативная C библиотека, потоковое API для больших объектов, ADO.NET, JDBC, ODBC

2.6. Sonic

Sonic [9] – это легковесный поисковый движок, который используется в SaaS платформе обмена сообщениями с клиентами для стартапов

и малого и среднего бизнеса «Crisp». Разработчики решили не использовать ElasticSearch, поскольку эмпирически выявили неприменимость этого поискового движка для своих задач.

Основными запрашиваемыми документами являются тексты сообщений, контакты, заметки и прочее, и у каждого клиента эти данные имеют разный формат – то есть данные являются плохо структурированными, что влечет за собой отсутствие общей схемы индексируемых данных. Реализация основных программных частей Sonic-а идеологически схожа с Lucene. Список проиндексированных слов хранится в отдельном сегменте, однако сегменты в Sonic изменяемы, в них можно добавлять и удалять слова. В Sonic по умолчанию есть простейшая реализация поэтапного поиска, который срабатывает в случае недостаточного количества найденных документов, и заключается в исправлении опечаток поискового запроса и повторного поиска с использованием уже альтернативных слов. Реализация исправления опечаток основана на конечном трансдьюсере (далее – конечный автомат). Например, для слова из 7 букв конечный автомат для нулевого расстояния Левенштейна содержит 8 узлов и 7 ребер, а то время как для расстояния 2 будет иметь 240 узлов и 753 ребра. В Lucene максимально возможным расстоянием для конечного автомата является расстояние в два символа. Это ограничение связано с экспоненциальным ростом объема данных, которые необходимо хранить, а также с увеличением времени построения конечного автомата. В Sonic также максимально возможным расстоянием Левенштейна является расстояние в два символа.

Далее приведены примеры построенных автоматонов для слова «Value» для различного расстояния Левенштейна.

Одной из особенностей Sonic является то, что результатом пользовательского запроса являются идентификаторы документа, из этого вытекает необходимость запросить у основной базы данных документы в исходном виде после выполнения поискового запроса. Это является следствием того, что метаинформация документа, а также проиндексированные поля не хранятся в поисковом сервере, что позволяет не тратить оперативную память на хранение исходных полей документа,

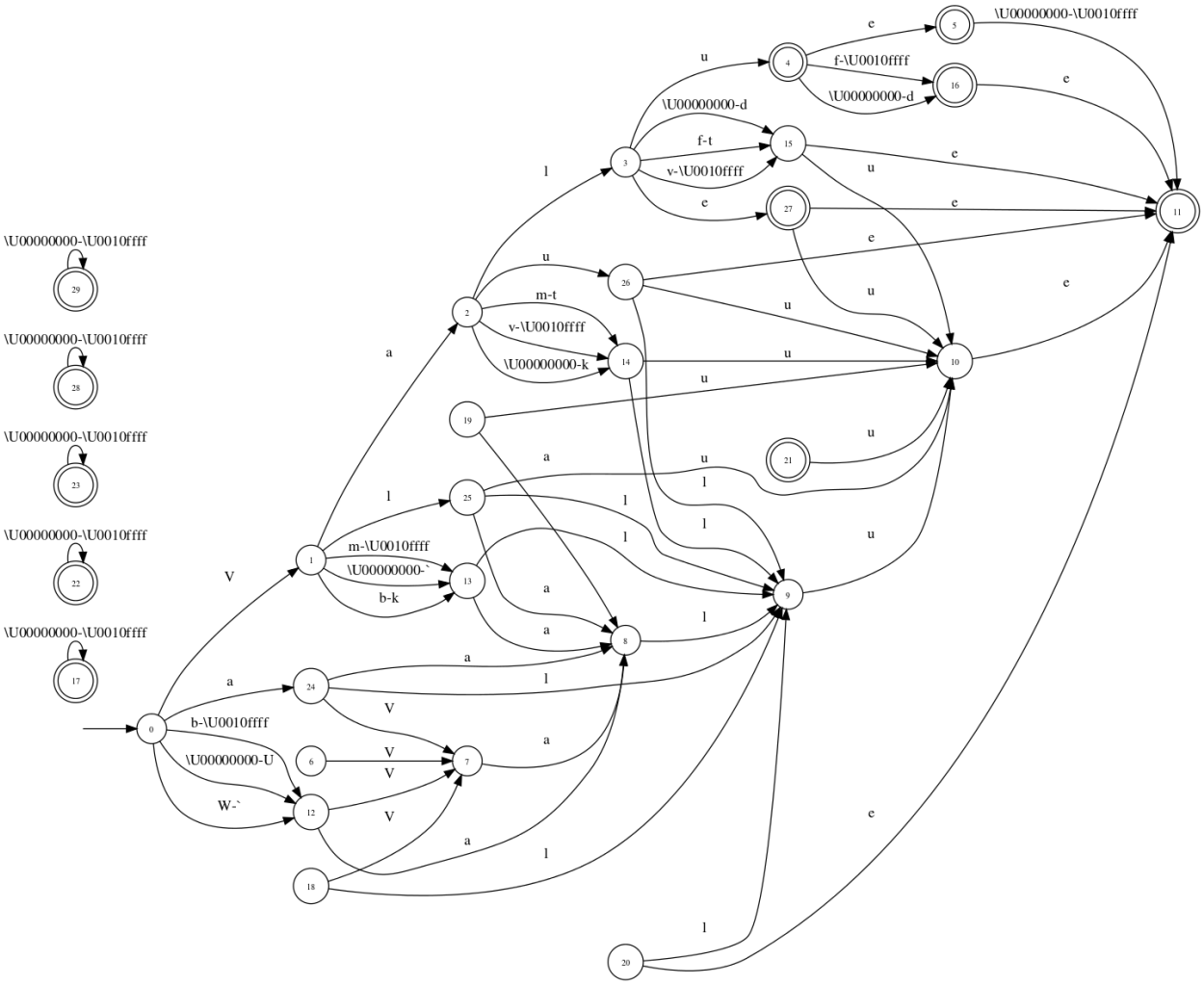

Рис. 2: Автоматон для расстояния Левенштейна 1

а также не заботиться о синхронизации данных.

Индексируемые слова хранятся в конечном автомате, который хранит для каждого слова ссылки на список соответствующих документов. Пересечение конечного автомата со списком всех слов, а также конечного автомата, построенного для слова поисковой фразы, является широко применимым способом нахождения релевантных документов. Этот метод также используется в Lucene, а также в Sphinx. В Lucene конечный автомат строится во время индексирования для каждого сегмента и обновляется только когда происходит слияние сегментов. В Sonic же конечный автомат перестраивается каждый раз, когда добавляется или убирается слово из индекса – это очень сильно усложняет систему, так как перестроение конечного автомата является трудозатратным про-

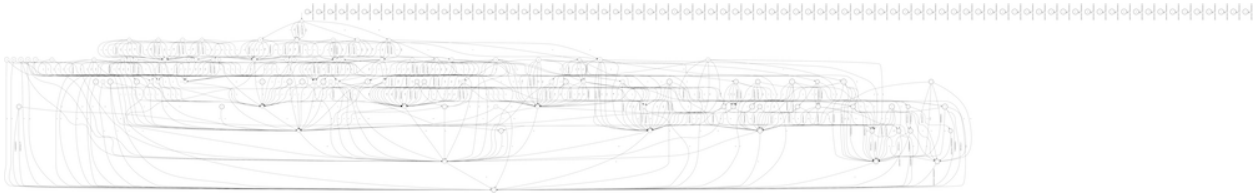

Рис. 3: Автоматон для расстояния Левенштейна 2

цессом. Перестроение конечного автомата также влечет за собой несогласованность данных, так как поиск может быть произведен до того как измененные данные будут видны.

- **Платформа:** C
- **Индекс:** инвертированный инкрементный индекс
- **Размер индекса и скорость индексации:** индекс занимает 20% от объема исходных данных
- **Типы документов:** текст
- **API:** NodeJS, PHP, Rust, Python, Ruby, Go, PHP, Java

2.7. Выводы

Из проведенного анализа нельзя сделать однозначный вывод о том, какая система наилучшим образом удовлетворяет требованиям, предъявляемым к компоненте поиска информации. Sphinx обладает весьма ограниченным функционалом для обработки текста в отличии от программных продуктах на базе Lucene, а также низкой скоростью обновления данных. Sonic также слишком ограничена по возможностям и применима только в небольших проектах с низкими требованиями по качеству поиска. PostgreSQL Fulltext Search требует большое количество памяти для хранения инвертированного индекса и не подходит для частого обновления полей документа. Полнотекстовый поиск в PostgreSQL весьма гибок, для него можно писать библиотеки при помощи официального API клиента на C++[3]. То есть доработка функционала, который есть по умолчанию в других программных продуктах

полнотекстового поиска, возможна, но это очень трудозатратно, и как следствие, трудно поддерживаемо. В статье «Об эффективности поиска данных в веб-приложениях»[12] сравнивалась эффективность выполнения полнотекстовых запросов с ранжированием в СУБД MySQL, PostgreSQL и Oracle. По результатам исследования были сделаны следующие выводы:

- При малом количестве данных рассматриваемые СУБД характеризуются схожими временными затратами при выполнении поисковых запросов
- СУБД PostgreSQL незначительно уступает СУБД Oracle и имеет большое преимущество перед СУБД MySQL

В статье «An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and Elasticsearch» 2016 года исследуются и анализируются различия между Solr и Elasticsearch, а также обсуждаются результаты производительности этих инструментов. Критериями анализа являются их скорости запросов и индексирования, простота и сложность использования, конфигурационные формы и архитектура. По результатом тестирования были сделаны следующие выводы:

- Solr и Elasticsearch являются аналогичными инструментами с точки зрения технических характеристик
- Solr использует меньше дискового пространства, учитывая размер данных после индексирования
- Что касается продолжительности индексирования, Elasticsearch показал лучшую производительность в коротких данных, в то время как Solr лучше показал себя на длинных данных
- Близкие результаты работы были получены во многих различных исследованиях

- Solr и Elasticsearch могут иметь совершенно разные характеристики производительности в определенных случаях
- Скорость QPS (query per second) может варьироваться в зависимости от типа данных
- Довольно трудно предсказать, какой инструмент будет иметь более высокую производительность.

Одно из сравнений Solr и ElasticSearch было проведено независимым Java консультантом Kelvin Tan[1], который рекомендует использовать Solr в случае если поиск является центральной частью продукта и UX, а также если приложение имеет конкретные и нюансные требования к релевантности поиска.

Основываясь на вышеуказанных статьях можно сделать вывод, что Solr является лучшим программным продуктом для использования в качестве информационно-поисковой системы для проекта "Юридическая аналитика".

3. Реализация

[15] Для написания системы был выбран язык Java, платформа полнотекстового поиска Solr и фреймворк для веб-приложения – Spring Framework. Solr запускается внутри отдельного контейнера сервлетов Spring.

Под Solr ядром необходимо понимать один инвертированный индекс, с соответствующими ему конфигурационными файлами и файлами логов. Всего было создано два ядра: основной – в котором хранятся индексируемые данные, и вспомогательный – в котором хранятся документы необходимые для концептного поиска. Схема документов хранится в файле «managed-schema» в каждом из ядер. В основном ядре наиболее важными полями являются «title» и «text», которые являются «stored» полями, то есть для них сохраняется изначальное представление.

В написании приложения использован MVCS (Model View Controller Service) паттерн. В сервисах описана бизнес-логика приложения, а также набор возможных операций для взаимодействия элементов системы.

Ключевым сервисом является «IndexServiceImpl», отвечающий за индексацию. После запуска приложения и инициализации ядер необходимо проиндексировать данные. Чтобы запустить индексацию основного ядра и концептного необходимо послать запросы на URL-ы. Индексируемые документы берутся из стороннего сервиса посредством дополнительного запроса, URL которого описан в файле «application.properties».

Поиск осуществляется посредством отправки GET запроса на URL «localhost:8080/search».

Параметры запроса:

- query – пользовательская фраза
- filter – список фильтров
- minResults – минимальное количество результатов
- maxResults – максимальное количество результатов

3.1. Реализация концептного поиска

Для концептного поиска используется отдельное ядро, в котором основными полями являются «searchTerm», в котором хранится название концепта, и «field», в котором хранится соответствующее концепту название поля. Во время построения экземпляра SolrQuery для дальнейшего поиска по основному ядру происходит поиск по всем словам в исходной фразе по концептному ядру. В результате поиска получается множество соответствий «field»:«searchTerm». Найденное слово убирается из фразы и добавляется в экземпляр SolrQuery в качестве фильтра. Данный подход можно использовать не только в качестве уточнения поля «title» или «text», в котором должно искаться слово, но и для создания фильтров по полям метаинформации.

3.2. Реализация поэтапного поиска

Поиск по основному индексу производится в два этапа. Для поиска по различным полям используется парсер запроса DisMax. Парсер DisMax используется для обработки запросов без сложного синтаксиса и поиска по многим полям, основываясь на весе каждого поля.

- На первом этапе по пришедшей фразе делается концептный поиск, результаты которого добавляются в экземпляр SolrQuery в качестве фильтров

- Далее, если количество найденных документов не превосходит заранее заданное значение, то производится проверка и исправление орфографических ошибок в пришедшей фразе, а затем повторяется первый этап с измененными орфографическими ошибками

Для проверки орфографии используется встроенный в Solr инструмент проверки орфографии – «SpellCheckComponent». В качестве инструмента проверки орфографии использован экземпляр «DirectSolrSpellChecker», который работает без построения отдельного индекса. В конфигурации проверки орфографии можно указать максимальное количество

исправлений, которое соответствует расстоянию Левенштейна. Как было указано в описании поискового сервиса Sonic максимально возможное количество исправлений – 2 символа. Результатом работы проверки орфографии является список в порядке количества вхождений в документы возможных исправлений для каждого из слов, которого нет в индексе. По умолчанию самое часто встречаемое слово считается исходным.

4. Интеграция в текущее приложение

На текущий момент весь проект «Юридическая аналитика» написан на Python[14], поэтому разработанный поисковый веб-сервис необходимо выделить в отдельный микросервис, и исходя из ограничений, указанных в техническом задании[16], запустить на той же машине, что и остальные сервисы. Поисковые запросы к Django необходимо перенаправлять к поисковому микросервису.

Перед индексацией данные, хранящиеся в Postgres, необходимо преобразовать в JSON формат и отправить запрос поисковому микросервису. Необходимо учесть, что ядра Solr непосредственно хранятся в оперативной памяти, поэтому при значительном увеличении объема данных необходимо будет решить вопросы репликации данных на несколько машин, а также усовершенствования поиска по кластеру микросервисов Solr.

5. Оценка качества поиска

Для оценки качества информационно-поисковой системы помимо метрик полноты и точности необходимо учитывать как в выбранном инструменте работает ранжирование результатов поиска. В информационном поиске используется понятие модели поиска. Модель поиска – это модель, на основании которой создается оценочная формула, позволяющая поисковой системе выбрать и отранжировать релевантные документы. Существует три вида моделей поиска: теоретико-множественная, алгебраическая и вероятностная. Lucene использует комбинированную модель поиска – для выбора документов используется теоретико-множественная (булева), а для ранжирования – алгебраическая (векторная). В подсчете веса документа принимают участие следующие факторы:

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost}() \cdot \text{norm}(t,d) \right)$$

Рис. 4: Формула расчета веса документа в Lucene

1. tf(t in d) – Частота, с которой термин появляется в документе. Учитывая поисковый запрос, чем выше частота, тем выше балл документа.

2. idf(t) – Обратная частота документа. Чем реже термин встречается во всех документах индекса, тем выше его вклад в оценку.

3. coord(q,d) – Координационный фактор. Чем больше терминов запроса найдено в документе, тем выше его оценка.

4. queryNorm(q) – Нормализующим фактор, используемый для сопоставления оценок между запросами. Этот фактор не влияет на ранжирование документов (поскольку все ранжированные документы умножаются на один и тот же фактор.

5. t.getBoost() – вес слова, указанный в поисковом запросе.

6. norm(t,d) – инкапсулирует в себе несколько факторов увеличения весов документа и полей, по которым производится поиск, которые добавлены при индексации.

Для оценки информационной поисковой системы была подготовлена коллекция документов, набор поисковых фраз и набор оценок релевантностей, представленных в виде бинарного значения – релевантный/нерелевантный. Для отображения результатов оценки был использован 11-точечный график полноты/точности, измеренный по методике TREC, который отражает изменение точности в зависимости от требований к полноте и дает более полную информацию, чем единая метрика в виде одной цифры. Помимо метрик полноты и точности методика TREC позволяет также учитывать ранжирование документов, полученных в результате оценки.

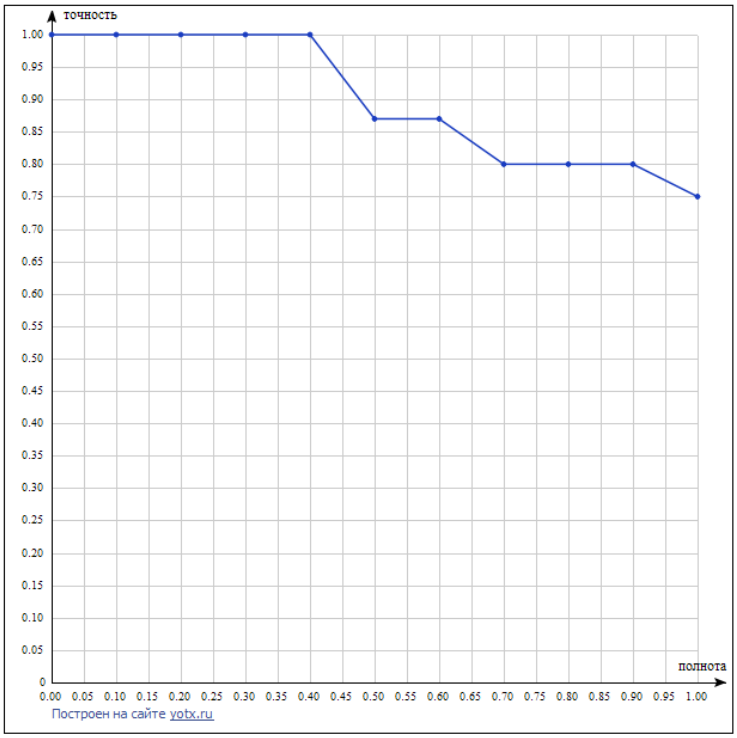

Рис. 5: 11-ти точечный график полноты/точности

Исходя из полученных результатов оценки работы информационно-поискового сервиса, можно сделать вывод, что разработанная система при максимальной полноте имеет хорошие показатели точности. Одна-

ко следует учесть, что мнение асессора может быть неточным, поэтому необходим более широкий анализ разработанного решения.

6. Заключение

В результате данной работы были проведены обзор и сравнение актуальных open-source поисковых систем, была изучена предметная область, предложены и рассмотрены подходы к улучшению метрик полноты и точности поискового запроса, а также реализовано Java-приложение.

Список литературы

[1] Ali AKCA Mustafa Aydogan Tuncay Ilkucar Muhammer. An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and Elasticsearch. International Journal of Intelligent Systems and Applications in Engineering. — 2016. — URL: https://www.researchgate.net/publication/311916747_An_Analysis_on_the_Comparison_of_the_Performance_and_Configuration_Features_of_Big_Data_Tools_Solr_and_Elasticsearch.

[2] Apache Solr vs Elasticsearch. — URL: http://solr-vs-elasticsearch.com/.

[3] The C++ connector for PostgreSQL. — URL: http://pqxx.org/development/libpqxx/.

[4] DB-Engines Ranking URL. — URL: https://db-engines.com/en/ranking/search+engine.

[5] ElasticSearch. — URL: https://www.elastic.co/.

[6] James H. Fowler Timothy R. Johnson James F. Spriggs II Sangick Jeon Paul J. Wahlbeck. Network Analysis and the Law: Measuring the Legal Importance of Precedents at the U.S. Supreme Court. — 2007. — URL: http://fowler.ucsd.edu/network_analysis_and_the_law.pdf.

[7] PostgreSQL full-text search. — URL: https://postgrespro.ru/docs/postgresql/9.5/textsearch.

[8] Solr. — URL: http://lucene.apache.org/solr/.

[9] Sonic – fast, lightweight and schema-less search backend URL:. — URL: https://github.com/valeriansaliou/sonic.

[10] Sphinx Search Engine. — URL: http://www.sphinxsearch.com/docs/sphinx3.html.

[11] Updating parts of documents in Solr. — URL: https://lucene.apache.org/solr/guide/6_6/updating-parts-of-documents.html.

[12] Земцов А.Н. Зунг Хань Чан. Об эффективности поиска данных в веб-приложениях // ИВД. №3 (46). — 2017. — URL: https://cyberleninka.ru/article/n/ob-effektivnosti-poiska-dannyh-v-veb-prilozheniyah.

[13] Лозовюк Александр. Полнотекстовый поиск в веб-проектах: Sphinx, Apache Lucene, Xapian. — 2008. — URL: https://habr.com/ru/post/30594/.

[14] Ссылка на проект «Юридическая аналитика». — URL: https://github.com/robot-lab/judyst-main-web-service/wiki/Additional-information.

[15] Ссылка на разработанную информационно-поисковую систему. — URL: https://github.com/ottenokLeshi/legaltech.

[16] Ссылка на техническое задание проекта «Юридическая аналитика». — URL: https://bit.ly/2HE9dLv.