Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки: 09.03.03 Прикладная информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка прототипа автоматизированной системы поиска дубликатов документов для цифровых научных библиотек

Работа завершена:

«___»_____2019 г.

Студент группы 11–502 _____М.В. Романов

Работа допущена к защите:

Научный руководитель

Доктор физ.–мат. наук, профессор

«___»_____2019 г. _____А.М. Елизаров

Директор Высшей школы ИТИС

«___»_____2019 г. _____А.Ф. Хасьянов

Казань – 2019 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент в мире существует огромное количество электронных научных библиотек, оперирующих большими объемами данных, в том числе и в текстовом формате. Количество научных документов неуклонно растет, и, чем больше их становится, тем сложнее отследить их уникальность. При добавлении новых документов необходимо проверять наличие их аналогов, что вручную выполнять очень долго и непрактично, особенно при больших объемах данных. В связи с этим возникает проблема проверки уникальности научных документов и нахождения их дубликатов в контексте научных библиотек.

Эта проблема часто решается посредством сервисов антиплагиата, которые оперируют своими закрытыми базами данных и, как правило, не дают к ним доступа. Но обособленные научные библиотеки, с быстро растущими объемами данных, не могут опираться на эти сервисы, как на безошибочный способ определения уникальности внутри самой библиотеки. Нет никакой уверенности в том, что научный документ, будучи занесенным в базу научной библиотеки, был добавлен и в базу сервисов антиплагиата. Поэтому для обеспечения уникальности документов научным библиотекам необходимо самим реализовывать систему поиска дубликатов, а это затратно как по времени, так и по силам.

Решением данной проблемы является создание сервиса поиска дубликатов научных библиотек с открытой и расширяемой базой данных, позволяющего научным библиотекам без лишних усилий вносить в него большие объемы данных и проверять на уникальность новые научные документы.

Объектом исследования является процесс создания сервиса поиска дубликатов научных библиотек.

Предметом исследования является поиск дубликатов документов научных библиотек.

Целью исследования является разработка сервиса поиска дубликатов в научных библиотеках.

Для достижения этой цели необходимо решить четыре задачи:

1. Исследовать способ организации электронных ресурсов научных библиотек с целью эффективного сбора данных научных документов;
2. Определить наилучший способ хранения данных научных библиотек с возможностью их расширения;
3. Исследовать существующие алгоритмы поиска дубликатов и определить лучше подходящий для большого объема постоянно изменяющихся данных;
4. Реализовать систему поиска дубликатов в научных библиотеках.

1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

1.1. Обзор литературы

Точное совпадение (четкие дубликаты). Пусть задана строка P, именуемая образцом, и более длинная строка T, именуемая текстом, тогда точным совпадением является множество всех вхождений образца P в текст T [1].

Неточное сопоставление – означает, что в принимаемом сопоставлении возможны ошибки разных типов, уточняемые в дальнейшем [1].

Операции редактирования – это вставка, удаление и подстановка или замена символов [1].

Редакционное расстояние – минимальное количество редакционных операций для преобразования одной строки в другую [1]. Иногда называется расстоянием Левенштейна.

Нечеткие дубликаты – схожие общие подстроки в двух документах. “Схожие” подстроки подразумевают собой неточное сопоставление, представленное редакционным расстоянием между этими подстроками. Проблема поиска дубликатов документов любой тематики, в том числе и научной, подразумевает под собой поиск нечетких дубликатов этих документов, потому что даже при малейшем изменении исходного текста, алгоритм точного определения дубликатов будет выдавать необъективный результат, утверждающий, что проверяемый и исходный тексты отличаются. Проблема обнаружения нечетких дубликатов является одной из наиболее важных и трудных задач анализа данных. Актуальность этой проблемы определяется множеством приложений, в которых нахождение нечетких дубликатов помогает решить определенные вопросы, например, улучшение качества индекса и архивов поисковых систем за счет удаления избыточной информации, фильтрация спама, выявление нарушения авторских прав. Сложность этой проблемы заключается в гигантских объемах данных,

хранимых в базах данных систем, в которых нужно проводить поиск дубликатов, что делает решение этой проблемы “в лоб” практически невозможным, точнее – очень затратным в плане производительности. Поэтому делается упор на снижение вычислительной сложности алгоритмов, избегая попарного сравнения документов [10].

Важными требованиями к алгоритмам определения нечеткого дублирования документов являются:

- Полнота – отношение количества правильно найденных дубликатов к реальному количеству дубликатов;
- Точность – отношение количества правильно найденных дубликатов к количеству неправильно найденных дубликатов;
- Устойчивость – устойчивость к маленьким изменениям проверяемого документа. Это означает, что результаты поиска дубликатов не должны меняться при незначительных изменениях документа.

Одной из первых работ в области определения нечетких дубликатов была работа Nevin Heintze [6], в которой он использовал нечеткие дубликаты для нахождения заимствований в текстах. Heintze реализовал поиск нечетких дубликатов на основе “отпечатков” каждого документа. Отпечаток состоит из всех возможных подстрок документа фиксированной длины, а его численное значение вычисляется по алгоритму случайных полиномов Карпа–Рабина [1]. В качестве меры сходства двух документов используется отношение числа общих подстрок к размеру файла или документа. Его работа устанавливает направление на снижения вычислительной сложности алгоритмов, избегая попарного сравнения всех документов и приводит пример использования “отпечатков” документов для их быстрого сравнения.

Новый метод, основанный на “синтаксической” оценке сходства документов, был предложен Andrei Broder [2]. Метод представляет документ в виде множества всех возможных последовательностей фиксированной длины,

состоящих из соседних слов. Такая последовательность называется “шингл”. Сходство документов определяется на основе пересечения их множеств шинглов. Чем больше выборка шинглов, тем более точными будут результаты. Но за эту точность приходится платить большими затратами на хранение этой выборки. Поэтому автором были предложены два метода сэмплирования для уменьшения выборки. Первый метод оставлял только те шинглы, чьи “отпечатки”, вычисляемые по алгоритму Карпа–Рабина [1], делились без остатка на некоторое число. Другой же метод отбирал только фиксированное число шинглов с наименьшими численными значениями “отпечатков” или оставлял все шинглы, если их общее число не превышало определенного значения.

Следующим шагом в развитии поиска нечетких дубликатов стала работа Dennis Fetterly [5]. Документ представляется множеством пятисловных “шинглов”. Затем вычисляется “отпечаток” каждого “шингла” с использованием алгоритма Карпа–Рабина [1]. Далее к каждому “отпечатку” применяются 84 различные (произвольно выбранные, но фиксированные) функции. Потом, для каждой функции выбирается “отпечаток”, минимизирующий значение соответствующей функции. Это приводит к тому, что документ представляет собой вектор из 84 “отпечатков”. После этого 84 шингла разбиваются на 6 групп по 14 шинглов, каждая такая группа называется “супершинглом”. Если два документа имеют сходство p ~ 0.95 (95%), то 2 соответствующих “супершингла” в них совпадают с вероятностью $p^{14} \sim 0.9514 \sim 0.49$ (49%). Так как каждый документ представляется 6 “супершинглами”, то вероятность того, что у двух документов совпадут не менее двух “супершинглов”, равна: $1 - (1-0.49)^6 - 6 * 0.49 \cdot (1-0.49)^5 \sim 0.90$ (90%). Из этого следует вывод, что для достаточно точного сравнения двух документов достаточно, чтобы у них совпадали не менее двух “супершинглов”, для этого строятся все возможные сочетания из 6 “супершинглов” по 2, каждое такое

сочетание называется “мегашингл”. Схожесть двух документов определяется сравнением их “мегашинглов”. Два документа являются схожими, если хотя бы 1 “мегашингл” у них совпадает.

IDF (inverse document frequency) – вес, определенный для каждого слова в коллекции. Вес определяется по формуле IDF = $log(N/n)$, где N – количество документов в коллекции, а n – количество документов, содержащих данное слово. Использование статистики коллекции IDF позволяет нам определить “полезность” терминов для обнаружения дубликатов документов [3, 7].

Следующим подходом, уже основанным на лексических принципах, стал алгоритм I–Match, предложенный Abdur Chowdhury [3, 7], использующий статистические данные коллекции и, по утверждению авторов, работающий в шесть раз быстрее, чем алгоритм “мегашинглов”. Основной идеей является построение I–Match “отпечатка” документа. Для исходной коллекции строится словарь слов, со средними значениями IDF, так как такие слова, как правило, обеспечивают более точные результаты при обнаружении нечетких дубликатов. Слова с большими и маленькими значениями IDF отбрасываются. Затем для каждого документа строится множество слов, входящих в него, и определяется пересечение этого множества и исходного словаря. Если размер этого пересечения больше некоторого минимального порога (определяемого экспериментально), то список слов, входящих в пересечение, упорядочивается, и для него вычисляется I–Match сигнатура (хеш–функция SHA1). Два документа считаются похожими, если у них совпадают I–Match сигнатуры. Проблема этого алгоритма была в том, что он был очень неустойчив к незначительным изменениям проверяемого документа, но этот недостаток был исправлен авторами путем добавления к начальному словарю множества словарей, построенных по начальному, в котором было удалено определенное количество слов, обычно 30%–35% от исходного объема, а сигнатуры документов строились по всех этим словарям, то есть строился вектор. Тогда

два документа считались похожими, если у них совпадала хотя бы одна из координат.

1.2. Имеющиеся решения

Sif – утилита для поиска всех похожих файлов в большой файловой системе. Скорость поиска всех групп похожих файлов, даже при сходстве всего лишь 25%, составляет от 500 МБ до 1 ГБ в час. Степень сходства и некоторые другие индивидуальные параметры могут быть определены пользователем. Sif также может быть использована для быстрой идентификации всех похожих файлов в запросе с использованием предварительно обработанного индекса. Sif находит свое применение в управлении файлами, сборе информации (для удаления дубликатов), повторном использовании программ, синхронизации файлов, сжатии данных и, возможно, даже в обнаружении плагиата [8].

KDF – это экспериментальная система для идентификации текстовых документов. Система ориентирована на документы по исследованиям в области компьютерных наук, такие, как технические доклады, документы конференций и журнальные статьи, однако существует и множество других применений. Целью системы KDF является отслеживание истории документа путем определения общих сегментов текста между документами. Типичные области применения системы включают в себя поиск цитат, отслеживание ссылок, отслеживание плагиата [6].

CCleaner, dupeGuru, SearchMyFiles и другие программы для поиска похожих или же идентичных файлов в операционной системе.

Сервисы антиплагиата, помогающие пользователям находить заимствования в курсовых работах, дипломах, диссертациях, научных статьях и отчетах.

2. ПРОБЛЕМЫ И ЗАДАЧИ

2.1. Организация данных научных библиотек

Проблема заключается в том, каким образом можно автоматически собирать данные научных библиотек, структура организации данных которых может варьироваться от одной к другой.

Для того чтобы иметь возможность собирать данные цифровых научных библиотек, необходимо знать их устройство и организацию. Как правило, цифровые научные библиотеки предоставляют доступ к тем или иным документам в виде “статей”, расположенных по определенному адресу. В лучшем случае для распределения адресов может быть использован RESTful API, когда каждая статья проиндексирована своим уникальным идентификатором в виде суррогатного ключа, то есть численным id, тогда проход по всем статьям библиотеки является простой задачей. В худшем–же случае статьи могут быть расположены по адресам, генерируемым по какому–то определенному шаблону, заданному самой электронной библиотекой, и задача обхода всех статей становится более затратной.

Рассмотрим организацию данных на примерах нескольких электронных научных библиотек:

Elibrary.ru – на главной странице представлены общее описание ресурса, полезные ссылки и статистика, отражающая текущее состояние научной библиотеки. Здесь также представлена навигация по сайту, через которую можно выйти на журналы, находящиеся в открытом доступе и статьи. Сами статьи представлены адресами в виде id: https://elibrary.ru/item.asp?id=27224423.

Habr.com – на главной странице отображены лучшие статьи за последние сутки, представлены полезные ссылки и общая навигация по сайту. В раздел со статьями можно перейти по кнопке “Публикации”, которая направляет вас на главную страницу сайта. Статьи представлены адресами в виде id: https://habr.com/ru/post/452584/.

Cyberleninka.ru – на главной странице представлены общее описание ресурса и его предназначение, популярные статьи и алфавитная навигация по статьям. Сами статьи представлены адресами в виде адреса, генерируемого по названию статьи, отображенному латиницей: https://cyberleninka.ru/article/n/rasshirenie–relyatsionnoy–algebry–rekursivnymi–strukturami.

Исходя из рассмотренных вариантов организации данных популярных научных библиотек, можно сделать вывод, что с главной страницы можно получить доступ к статьям, содержащимся в научных библиотеках, но шаблон адресов статей не всегда предсказуем, что затрудняет автоматическую навигацию по статьям.

2.2. Способы хранения данных

Для того чтобы иметь возможность находить дубликаты документов научных библиотек, необходимо иметь определенную базу данных, в которой будет производиться поиск. Проблема заключается в том, что научные библиотеки оперируют огромным количеством данных, поэтому остро стоит задача определения оптимальной организации данных внутри системы, которая позволяла бы быстро добавлять данные и производить по ним поиск и сравнение.

2.3. Требования к системе

Основные требования к системе:

- Оптимальный по времени поиск дубликатов документов;
- Возможность добавления в базу системы данных научных библиотек;
- Возможность расширение базы системы новыми данными;
- Быстрый обход документов цифровых научных библиотек с целью их индексации.

Система должна представлять собой общедоступный сервис для проверки дубликатов документов научных библиотек и функционировать следующим образом:

- Пользователь сервиса добавляет адрес определенной электронной научной библиотеки в базу сервиса;
- Сервис автоматически индексирует все документы этой электронной научной библиотеки;
- Пользователь загружает текст проверяемого документа в сервис и выбирает из списка проиндексированных научных библиотек ту, в которой необходимо сделать проверку на дублирование;
- Сервис проверяет на дублирование и выдает результат в виде информации о том, в каких документах имеется дублирование, либо о том, что его нет.

Иные требования к системе:

- Пользователь должен иметь возможность удаления электронной библиотеки из очереди индексируемых библиотек без потери уже проиндексированных данных;
- Должна выводиться статистика для проиндексированных библиотек в виде количества содержащихся в них документов, их адресов и даты последней индексации библиотеки;
- Пользователь должен иметь информацию о том, сколько библиотек поставлено в очередь на индексацию, какая библиотека индексируется в данный момент и на каком этапе находится индексация;
- Проверка документов на дубликаты и пересчет значимости слов должны выполняться за приемлемое время.

3. ПОДХОД И РЕШЕНИЕ

3.1. Обход электронного ресурса

За основу успешного обхода ресурса берется тот факт, что с его главной страницы возможно попасть в секцию со статьями и получить адреса статей. Для сбора данных электронной библиотеки используется рекурсивный обход всех ссылок в пределах электронной библиотеки. Начальный набор ссылок берется с главной страницы (Рис. 1).

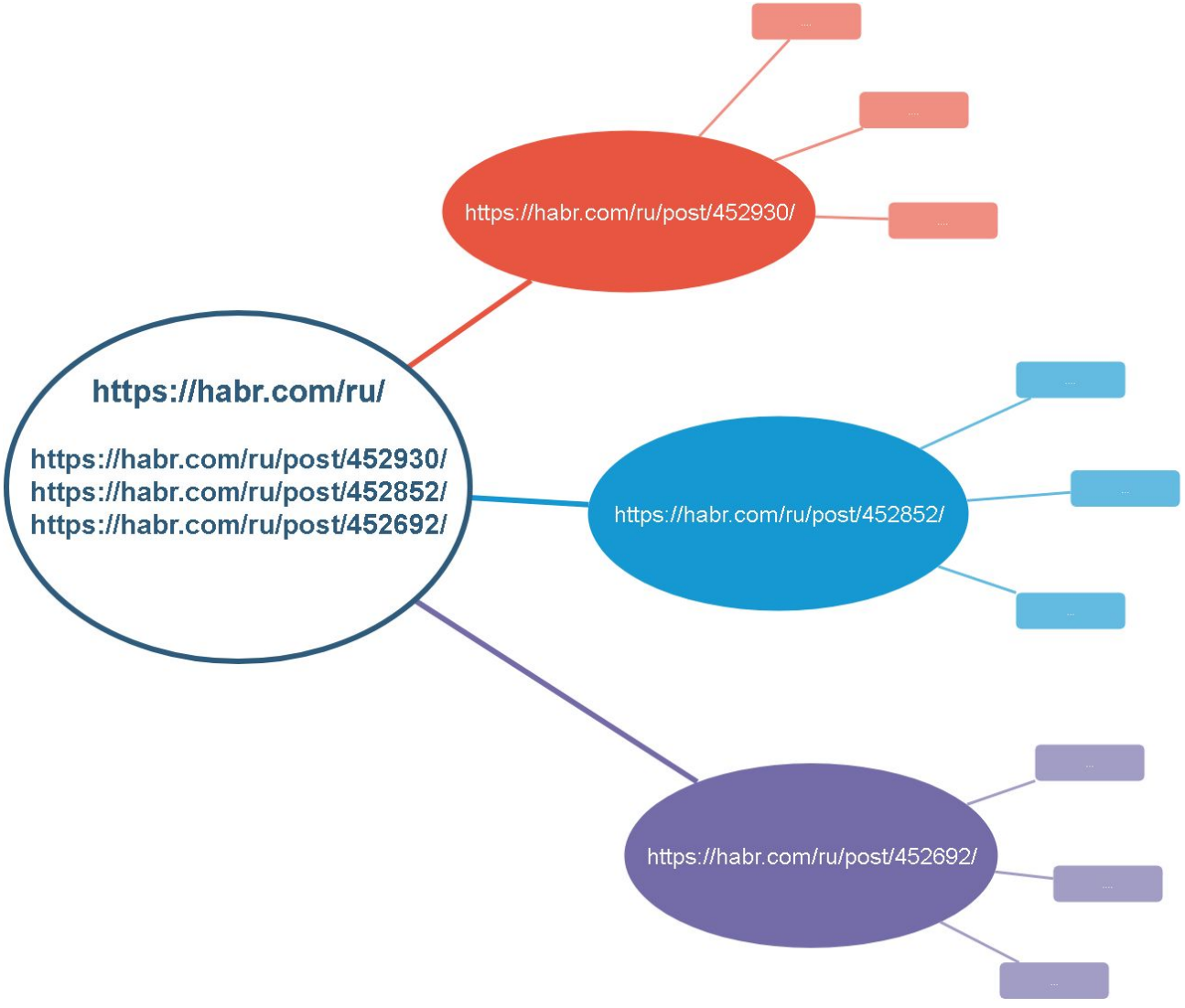

Рис. 1. Сбор ссылок

При таком подходе количество документов, добавляемое в базу данных, является слишком большим и не репрезентативным, потому что содержит в себе не только актуальные статьи, но и стороннюю информацию, как,

например, профили пользователей сайта, новости, различные форумы и т.д. Для решения этой проблемы решено было ввести систему “уточнений”, определяемых ключевыми словами, которые указывают, по какому адресу примерно содержатся нужные документы. Например, пользователь может с уверенностью сказать, что все необходимые документы содержат в адресе строку “post”, тогда строка “post” является уточнением и сервис, при добавлении документов , проверяет, содержит ли их адрес строку “post”. Таким образом удается достичь ограничения количества добавляемых документов и сохранить их репрезентативность.

Самым первым этапом является задание пользователем адреса главной страницы электронной библиотеки. Внутри сервиса определены конкретные этапы индексирования библиотеки. После задания адреса библиотеки сервис переходит в состояние “поиск ссылок”. На этом этапе сервис собирает все ссылки с главной страницы, далее происходит рекурсивный обход этих ссылок с целью поиска адресов, содержащих строки уточнения. Это происходит до тех пор, пока не будет найдено десять адресов, удовлетворяющих условию.

Десять найденных адресов, удовлетворяющих условию уточнений, теперь будут использованы, чтобы найти закономерность в распределении адресов статей либо понять, что ее нет и научная библиотека использует свой шаблон для распределения адресов статей. Данная закономерность необходима для того, чтобы далее не использовать рекурсивный обход по ссылкам. Хоть этот подход и является рабочим, он затратен по времени, так как нельзя точно определить рамки обхода, и количество найденных на сайте ссылок может быть просто запредельным. Закономерность в распределении адресов помогает разграничить четкий диапазон ссылок, которые нужно пройти, чтобы внести в систему необходимые данные.

После того, как информация о закономерности распределения адресов либо ее отсутствии получена, сервис переходит в режим “в работе”. На данном

этапе происходит сбор информации статей. Если закономерность отсутствует, сервис выполняет рекурсивный обход всех ссылок, начиная с главной страницы собирая содержимое страниц. Если же закономерность была найдена, то на ее основе строится диапазон ссылок, которые необходимо обойти, чтобы собрать данные статей. Далее ссылки из диапазона обходятся, и собирается содержимое страниц.

Отдельным случаем является обход библиотеки, содержимое которой частично уже есть в базе данных сервиса. Это представляет собой отдельный функционал, потому что перед обработкой страниц сервис проверяет, не были ли обработаны эти страницы ранее, а если библиотека ранее частично была занесена в базу данных, то эта проверка является еще более критичной, так как помогает избежать лишних обращений к базе данных и избыточности данных. Для организации этой проверки из базы данных сервиса достаются адреса всех документов, которые уже были проиндексированы. Можно смоделировать ситуацию, когда все возможные ссылки с главной страницы уже были собраны, обработаны и эти страницы были добавлены в базу данных. Тогда сервис, начиная с главной страницы, попадает в тупик, так как больше нет ссылок, которые он еще не обошел. В таком случае, даже если в библиотеке содержится более полумиллиона документов, а было обработано только пару сотен, сервис все равно может решить, что документов больше нет, так как пути к другим документам он найти не сможет. Для решения этой проблемы был придуман способ “рукопожатий”, при использовании которого сервис игнорирует проверку на избыточность данных до того момента, как он не найдет определенное количество новых ссылок, с которых он может продолжить путь к новым документам. Даже при применении этого способа проверка на избыточность данных не проводится только в функционале сбора ссылок, избыточные данные в любом случае не добавляются в базу, так как предварительно при записи проводится проверка на уникальность. Таким

образом получается избежать тупиковой ситуации с отсутствием путей к новым документам.

Ранее уже упоминалось про диапазоны ссылок, которые строятся на основе шаблона адреса расположения документов. При использовании диапазонов поиск ссылок необходим только на этапе вычисления шаблона, далее же идет простой перебор всех ссылок из диапазона. В таком случае при частичной индексации библиотеки необходимо запоминать как сам диапазон ссылок, так и последнюю обработанную страницу и, при повторной индексации библиотеки, доставать эти значения из базы данных и продолжать индексацию, начиная с последней обработанной страницы.

Все эти решения помогают быстро производить обход электронных ресурсов научных библиотек, избегая обработки “лишних” страниц, используя диапазоны ссылок и уточнения. Благодаря же системе “рукопожатий” и запоминания диапазонов и последних обработанных страниц, получается достичь большей гибкости в частичной индексации и повторной индексации цифровых научных библиотек.

3.2. TF–RIDF и веса слов

Алгоритмы поиска дубликатов делятся на два типа: синтаксические и лексические, к синтаксическим относятся все алгоритмы, связанные с шинглами, последовательностями соседних букв или слов, к лексическим относятся алгоритмы, основывающиеся на словах как на элементарных единицах, представителем таких алгоритмов является I–Match. В данной работе выбор алгоритма пал на TF–RIDF, являющийся лексическим, глобальным алгоритмом, учитывающим общую статистику коллекции и опирающимся на частотные характеристики слов.

Основная идея RIDF (Residual IDF) [4] состоит в сравнении двух способов подсчета количества информации (в смысле определения К. Шеннона), содержащейся в сообщении о том, что данное слово входит в некоторый

документ (по меньшей мере, один раз). Первый способ, статистический, это обычный IDF = $-log(df/N)$, где df – число документов в коллекции, содержащих данное слово, а N – число документов в коллекции. Второй способ, теоретический, основан на модели распределения Пуассона, предполагающей, что слова в коллекции документов распределяются случайным и независимым образом, равномерно рассеиваясь с некоторой средней плотностью. В этом случае соответствующее количество информации равно P_IDF = $-log(1 - exp(-cf/N))$, где cf – суммарная частота слова в коллекции, а N – число документов в коллекции. RIDF = $IDF - P\_IDF$ и показывает прирост информации, содержащейся в реальном распределении слова в коллекции по сравнению с равномерно случайным пуассоновским, т.е. ценность слова. Другими словами, «хорошие» (значимые, осмысленные) слова должны быть распределены неравномерно среди относительно небольшого числа документов (обладать «редкостью»), а «плохие» (бессодержательные) будут равномерно рассеяны по всей коллекции (встречаться, что называется, «на каждом шагу»).

Веса слов определяются по формуле w = $TF * RIDF$. TF = $0.5 + 0.5 * tf / tf\_max$, где tf — частота слова в документе, а tf_max — максимальная частота слова в документе.

Из всего сказанного выше ясно, что алгоритм TF–RIDF опирается как на “локальные” данные, собранные только в контексте одного документа, такие, как TF, и на “глобальные” данные, собирающиеся на основе статистики всей коллекции.

Данный алгоритм был выбран по нескольким причинам, первой из которых являются его хорошие результаты в исследовании, проведенном в работе [10], где его плотность составила 0.59, а точность 0,95. Второй аргумент в пользу этого алгоритма – это то, что он опирается на общую статистику коллекции, что снижает вычислительные расходы для непосредственно самой проверки на дублирование документов. Третьим и последним плюсом этого

алгоритма является его направленность на равномерное распределение “плохих” слов в документе, что помогает фильтровать нерепрезентативные слова и выделять из всего набора слов только те, что имеют реальную ценность. Для данной системы это свойство является критичным, потому что сбор информации ведется с электронных ресурсов, и не всегда возможно заранее отфильтровать “плохие” слова из выборки.

3.3. Архитектура системы

Архитектура системы представляет собой web–интерфейс взаимодействия с пользователем и серверную часть. С помощью web–интерфейса возможно точное управление процессом индексации цифровых научных библиотек и передача серверу документов для проверки на дубликаты.

Пользователь определяет, какую цифровую научную библиотеку он добавит в очередь на индексацию, сервер добавляет ее в очередь и индексирует.

Пользователь выбирает из списка уже проиндексированных библиотек, в какой он хочет проводить поиск на дубликаты документов, и добавляет текст проверяемого документа. На сервере проверяемый документ проходит определенные ниже этапы:

- Проводится синтаксический анализ текста, из него выделяются все слова;
- Слова проходят стемминг, выделяются основы слов, чтобы привести слова, имеющие одинаковый смысл, но различающиеся написанием, к одному виду;
- Из текста удаляются стоп–слова, определенные заранее, так как они не несут смысловой нагрузки;
- Удаляются слова, длиной короче трех символов, по той же причине, что и стоп–слова.

После прохождения всех этапов предобработки текст приведен к общему виду и готов к проверке на дубликаты:

- Считается общее количество слов;
- Для всех слов рассчитывается их частота в документе и TF;
- Для каждого слова из базы данных достаются количество документов, содержащих это слово, общее количество этого слова в коллекции, на основе этих значений вычисляются IDF и RIDF;
- Далее для каждого слова вычисляется его значимость, а именно – значение TF–RIDF;
- Из документа извлекаются шесть слов с наибольшим значением TF–RIDF, сортируются в алфавитном порядке и склеиваются в одну строку, контрольная сумма которой является сигнатурой документа;
- Эти же действия проводятся для всех документов коллекции;
- Проводится проверка на схожесть документов путем сравнения их контрольных сумм.

Структура базы данных определяется пятью основными таблицами (Рис. 2).

Таблица library описывает научную библиотеку, ее адрес, время последней индексации, общее количество слов, собранных из научной библиотеки. Таблица word определяет слова, собранные в определенную библиотеку. Слова можно было бы сделать независимыми от библиотек, так как одни и те же слова могут повторяться в разных библиотеках, это бы уменьшило общий объем хранимых данных, но для разных библиотек значения общего количество слова в коллекции и количества документов с вхождениями этого слова различны, поэтому необходимо обособлять слова в контексте библиотек, для быстрого пересчета значимости слов.

Clarification – уточнения, введенные пользователем, вносящим библиотеку в базу данных сервиса, обозначают, где примерно нужно искать документы внутри электронного ресурса библиотеки.

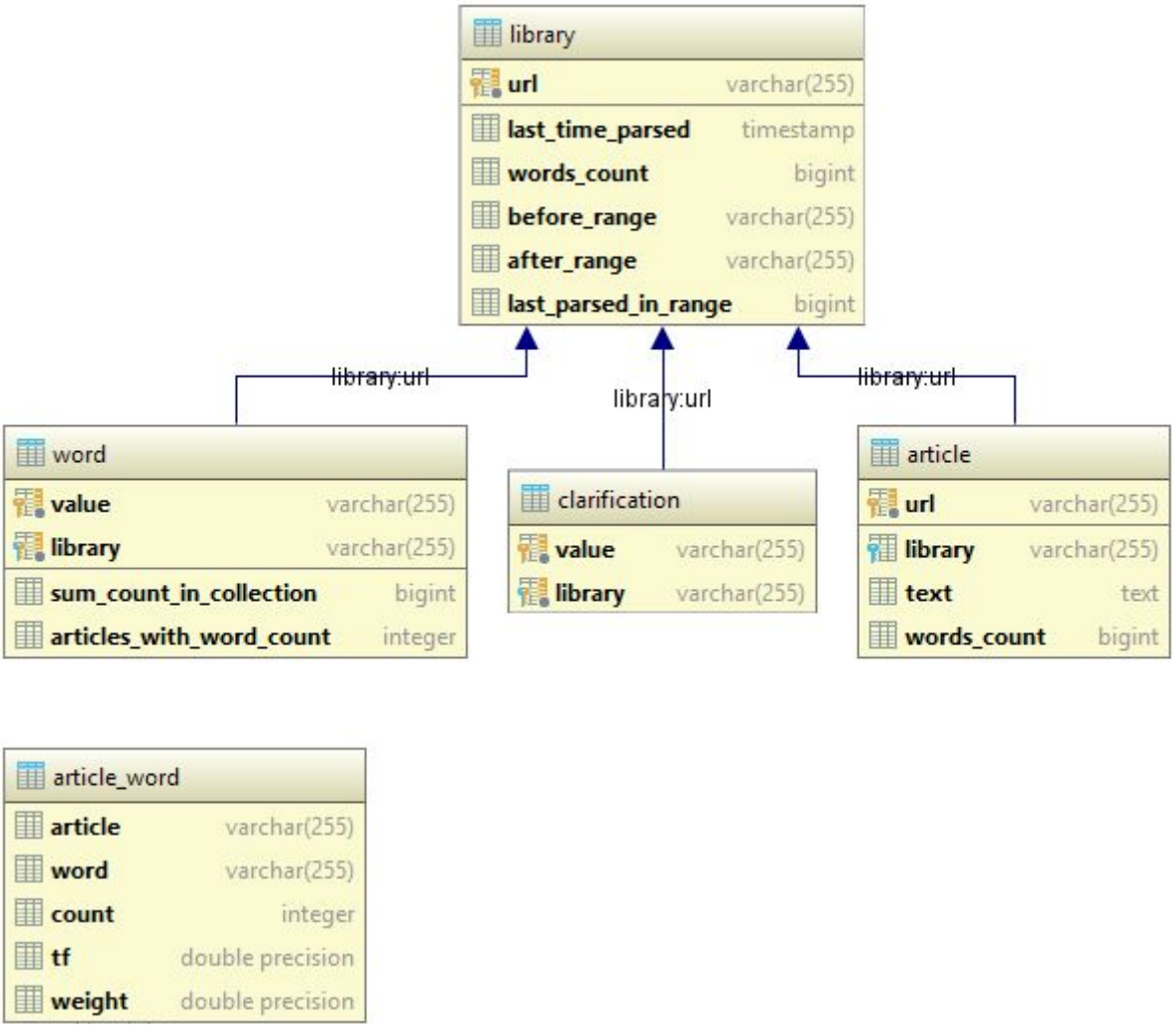

Рис. 2. Диаграмма базы данных

Article – документы, содержащиеся в библиотеке, их содержание с количество слов. Основной таблицей является article_word, она описывает связь между словами и документами, именно в этой таблице хранится TF и значимость слова в контексте документа. Как можно заметить, article_word не имеет внешних ключей, это сделано для того, чтобы обновление значения TF–RIDF проходило быстрее, так как при наличии внешних ключей база данных проверяет условия соответствия внешнему ключу, а это лишнее время.

Так как численное значение важности слова в контексте документа вычисляется на основе статистики всей коллекции, его невозможно вычислить “локально” – не прибегая к базе данных, но если бы для каждого слова приходилось делать обращение к базе данных, это заняло бы колоссальное

количество времени и ресурсов. Поэтому было принято решение при добавлении документа вычислять все его локальные значения, которые только можно вычислить и заносить их в базу данных большими группами, в одном запросе. Само же вычисление значимости слов было вынесено в базу данных в виде процедуры, которая выполняется после того, как все документы их локальными значениями были внесены в базу. Процедура осуществляет вычисление значимости всех слов в коллекции на основе их локальных данных и статистики всей коллекции, которая собирается при внесении документов. Также такой подход обеспечивает гибкость в добавлении и обновлении данных. Если бы важность слов вычислялась при добавлении документов, отпадала бы возможность добавлять новые документы в коллекции, так как пришлось бы заново собирать всю статистику коллекции, которая была уже собрана, повторно рассчитывать значимость всех слов, удалять старые значения и добавлять те же данные, с измененной статистикой. Вместо этого документы и слова добавляются с локальными данными, а на основе общей статистики коллекции рассчитывается значимость слов. При таком подходе добавление новых документов не представляет новые данные: новые документы добавляются в базу с рассчитанными локальными данными, одновременно с этим обновляется статистика коллекции – дополняется новыми документами. Далее у всех слов в коллекции пересчитывается значение TF–RIDF на основе новых данных.

4. РЕАЛИЗАЦИЯ

4.1. Технологии

Для реализации системы был выбран язык программирования Java 8, с использованием фреймворка Spring Boot для быстрой настройки приложения и простой реализации серверной части.

Быстрое описание моделей и методов доступа были реализованы с помощью библиотеки Lombok, которая посредством аннотаций позволяет избежать лишнего шаблонного кода.

Организация многопоточности и агрегации данных были достигнуты с помощью таких возможностей Java 8, как concurrent и потоки.

Для хранения данных была выбрана СУБД PostgreSQL, так как она является оптимальным и, в то же время, бесплатным вариантом реляционной базы данных.

Для реализации потокобезопасных коллекций были выбраны средства concurrent и библиотеки Guava.

Доступ к базе данных из кода был реализован посредством библиотеки Spring–jdbc, позволяющей писать как простые, так и сложные запросы к базе данных, забирая при этом на себя ответственность за маппинг сущностей.

Стемминг проводился средствами библиотеки Mystem–scala, предоставляющей множество возможностей для анализа семантики слов.

Сбор html–страниц с электронных ресурсов научных библиотек организован посредством использования библиотеки Jsoup.

4.2. Многопоточная обработка ресурса

При внесении пользователем цифровой научной библиотеки в очередь на индексацию, она добавляется в коллекцию, реализацией которой является объект класса ConcurrentLinkedQueue из библиотеки Guava. ConcurrentLinkedQueue позволяет организовывать потокобезопасный, а, в данном случае, еще и последовательный доступ к коллекциям. При запуске

системы создается и начинает свое исполнение отдельный поток – диспетчер (Рис. 3), который отвечает за диспетчеризацию библиотек, добавленных в очередь на обработку. Поток постоянно проверяет состояние верхней в очереди библиотеки. Для новых библиотек он запускает процесс их обработки. Если библиотека находится в состоянии обработку, поток не выполняет никаких действий и ожидает завершения процесса обработки. Когда библиотека переходит в завершенный статус, поток удаляет ее из очереди на обработку, достает верхний элемент и запускает процесс обработки.

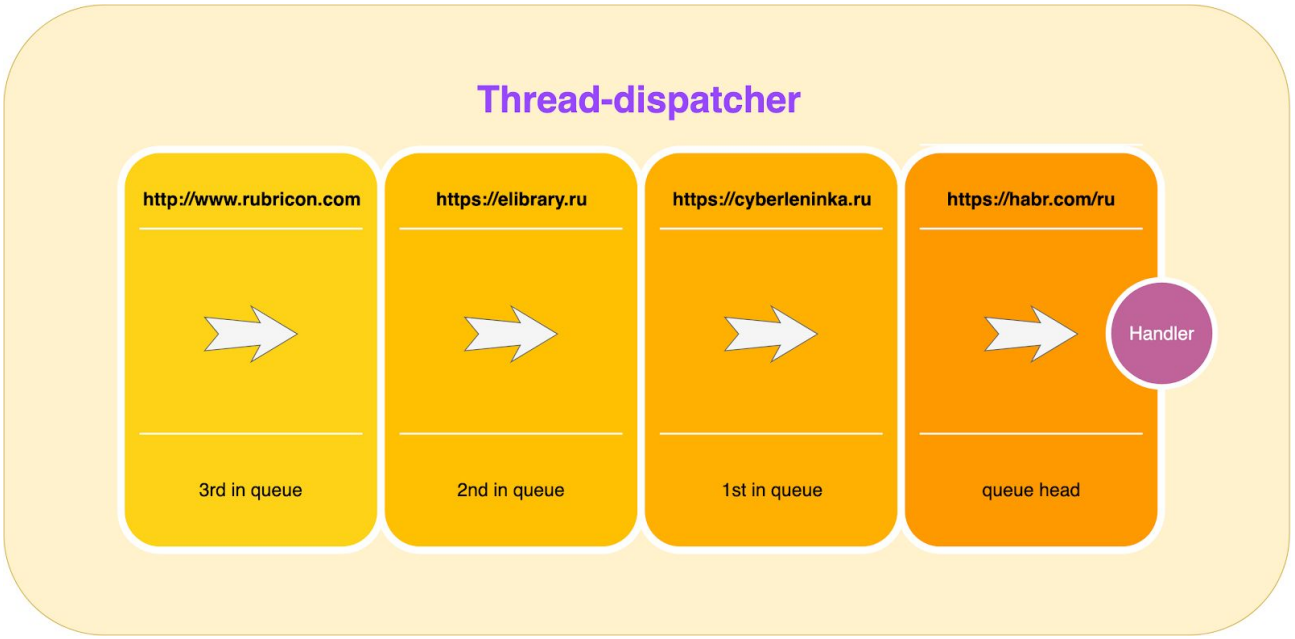

Рис. 3. Очередь библиотек на обработку

После того, как диспетчер запускает обработку библиотеки, осуществляется проверка наличия этой библиотеки в базе данных системы. Если библиотека отсутствует в базе данных, она туда добавляется и для нее запускается обработчик, проводящий анализ ссылок, поиск шаблона распределения адресов, вычисление диапазона ссылок для обработки, парсинг уточнений. Если же библиотека была ранее добавлена в базу данных системы, то достаются все адреса уже обработанных ссылок, последняя обработанная ссылка, строится диапазон обхода и запускается обработчик, которые, на

основе уже проведенного анализа электронного ресурса, выполняет обход всех оставшихся ссылок из диапазона.

Обработчик использует многопоточную модель для ссылочного обхода электронного ресурса библиотеки. Многопоточность используется для обхода, потому что время, затрачиваемое на доступ к электронному ресурсу, в разы превышает время обработки самой страницы и вычисление локальных данных документа, поэтому оптимально организовывать доступ к страницам библиотеки в несколько параллельных потоков. Многопоточный доступ к ресурсу обеспечен средствами Java 8 concurrent. Все ссылки, которые необходимо обработать, помещаются в потокобезопасную коллекцию ConcurrentHashMap для одновременного доступа к ней из нескольких потоков. Изначально в отображении находится только ссылка на главную страницу ресурса, после того, как все ссылки с главной страницы были собраны и отфильтрованы, они также помещаются в отображение, и для каждой ссылки запускается отдельный поток для обработки, который собирает все ссылки со страницы, после того, как ссылки были собраны, запускается поток, который проводит синтаксический анализ текста со страницы, рассчитывает локальные данные и добавляет все слова и сам документ в базу данных.

Организация потоков осуществляется средствами ExecutorService. Сервис позволяет распределять задачи между различными потоками, управлять загрузкой системы и успешно диспетчеризировать исполняемые задачи. Так как для записи в базу данных необходимо больше времени, для этого было выделено наибольшее число потоков – 5. Для доступа к электронному ресурсу изначально планировалось выделить 5 потоков, но тестирование системы показало, что 5 потоков для этой цели будет слишком много, так как потоки, пишущие в базу данных, не успевают обрабатывать все те страницы, которые собирают потоки–сборщики. Поэтому решено было оставить сборщикам только 2 потока. Для сбора всех ссылок со страниц был выделен 1 поток, так как это

очень быстрая операция, не требующая доступа к каким–либо удаленным ресурсам.

Для успешной организации взаимодействия между потоками обработчика каждой странице в отображении присваивается ее текущий статус в жизненном цикле обработки. Сразу после ее добавления в отображение она имеет статус “новый”. “Новые” страницы подхватываются потоком–сборщиком, который собирает содержимое с электронного–ресурса, как только поток–сборщик захватил страницу для ее последующей обработки, он устанавливает ей статус “зарезервировано” для того, чтобы другие потоки не забирали эту же страницу. После успешной сборки содержимого страницы, ей устанавливается статус “проверена” и она подхватывается потоком, собирающим все ссылки и добавляющим эти ссылки в отображение со статусом “новый”, статус страницы меняется на “собранная”. Далее страница резервируется потоком–парсером, который осуществляет парсинг страницы, собирает слова, вычисляет локальные значения и добавляет документ в базу данных.

Работа обработчика прекращается либо по команде извне, либо по окончании ссылок на обработку. Как только обработчик завершает свое исполнение, для данных, которые он добавил, вызывается метод пересчета важности слов.

4.3. Добавление и обновление данных

Так как система содержит в себе огромное количество данных, необходимо правильно организовать добавление данных и их обновление. Основным источником проблем здесь становятся две таблицы, а именно, word и aritcle_word, так как содержат наибольшее количество записей. Если для каждого слова вызывать обращение к базе данных, добавление будет происходить непозволимо долго. Поэтому для добавления огромного количества слов была использована технология batch update, позволяющая строить один большой запрос к базе данных, добавляющий или обновляющий

множество записей за одно обращение. Таким образом, все запросы, добавляющие слова в базу данных, инициируют один запрос, представляющий из себя “пачку запросов”; так получается избежать множественного обращения к базе данных. Эта технология также позволяет контролировать и размер самой пачки.

У каждого слова есть параметры, которые могут потенциально меняться при добавлении новых документов, такие, как количество документов, содержащих это слово и общее количество этого слова в коллекции. Необходимо учитывать возможность изменения значения этих параметров при добавлении новых документов. Это достигается аналогом команды MERGE из Oracle, добавленной в PostgreSQL с версией 9.5. Команда называется INSERT ON CONFLICT и позволяет вставлять данные в базу и, при нарушении определенного условия уникальности данных, производить определенные действия с данными. Эта команда используется для вставки новых слов в базу данных: если этого слова еще нет в базе, выполняется простой INSERT, если же уникальность слова нарушается, выполняется UPDATE, который изменяет параметры articles_with_word_count и sum_count_in_collection. Таким образом происходит поддержание статистики коллекции в актуальном состоянии средствами самой базы данных.

Хоть и проблемы, описанные выше, являются довольно важными, самой актуальной проблемой является проблема обновления огромного количества записей, для высчитывания значимости слов. PostgreSQL позволяет проводить UPDATE со скоростью 1500 обновлений в секунду, что является внушительным числом, но не для базы, содержащей несколько миллионов записей для каждой коллекции. Если база будет содержать 5000000 записей, то процесс обновления этих записей займет около часа, что неприемлемо долго. Для ускорения работы UPDATE было предпринято несколько решений. Для начала были удалены все ограничения с таблицы article_word, проверка

который замедляла работу обновления. Следующим шагам стали создание для каждого обновления временной таблицы и вставка в нее вычисленных значений важности слов. Это было предпринято, потому что вставка во временную таблицу и выполнение операции UPDATE FROM выполняются в разы быстрее, чем простой UPDATE. Скорость выполнения операции обновления повысилась с 1500 о/с до 18000 о/с. Теперь 5000000 записей обновлялось примерно за 4,5 минуты. Гораздо быстрее, чем простое обновление, но недостаточно быстро. Последней оптимизацией стало регулярное осуществление операции VACUUM, которая очищает базу данных от “мертвых” кортежей. При обычных операциях кортежи, удалённые или устаревшие в результате обновления, физически не удаляются из таблицы, они сохраняются в ней, пока не будет выполнена команда VACUUM. Таким образом, периодически необходимо выполнять VACUUM, особенно для часто изменяемых таблиц. Она высвобождает пространство и делает его доступным для повторного использования [9]. После всех выполненных решений по оптимизации обновления стали выполняться со скоростью ~100000 о/с, а 5000000 записей обновляются примерно за 1 минуту, что приемлемо, так как операция обновления выполняется лишь один раз, после полной обработки данных библиотеки.

Для выполнения обновлений была создана процедура, вызываемая для определенной научной библиотеки, которая пересчитывает значимость всех слов в библиотеке. Процедура находит все документы, содержащиеся в библиотеке, создает временную таблицу, если ее еще нет, выполняет в ней операцию TRUNCATE, добавляет в нее все значения TF–RIDF, рассчитываемые по формуле. После этих действий она производит операцию UPDATE FROM и обновляет значимость всех слов в коллекции, доставая численное значение значимости из временной таблицы. Процедура запускается

каждый раз после добавления данных новой библиотеки, либо при расширении данными старой библиотеки.

4.4. Контрольная сумма и дубликаты

Каждый документ имеет свою уникальную сигнатуру, называемую также контрольной суммой. В общем виде контрольная сумма представляет собой некоторое значение, вычисленное по определённой схеме на основе кодируемого сообщения. Для сравнения двух документов необходимо сравнивать их сигнатуры. Сигнатура вычисляется на основе CRC32 – алгоритма вычисления контрольной суммы – способа цифровой идентификации некоторой последовательности данных, который заключается в вычислении контрольного значения её циклического избыточного кода. Входным значением для CRC32 является строка, составленная по следующему принципу:

- Из документа выбираются 6 слов, с наибольшими значениями TF–RIDF;
- Слова сортируются по алфавиту в порядке возрастания;
- Склеиваются в одну строку.

Для проверки входящего документа на дубликаты внутри определенной библиотеки необходимо сначала рассчитать его контрольную сумму и потом сравнить с контрольными суммами документов из базы данных. Документ проходит этап парсинга, стемминга, сбора слов. Для каждого слова высчитывается его частота внутри документа по формуле: tf = $word\_count / article\_words\_count$, где word_count – количество повторений слова в документе, article_words_count – общее количество слов в документе. На основе частоты вычисляется TF, по формуле, приведенной в разделе 3.2, tf_max – максимальная частота в документе, определяется после расчета всех частот в документе. Для проверяемого документа необходимо вычислить IDF и RIDF. Для этой цели все каждому слову проверяемого документа сопоставляется сущность этого слова из базы данных, с рассчитанными

глобальными значениями. На основе этих глобальных значений, таких, как число документов в коллекции, число документов, содержащих данное слово, суммарная частота слова в коллекции, вычисляется значимость всех слов проверяемого документа. По алгоритму, приведенному выше, составляется сигнатурная строка документа, подается на вход алгоритму CRC32. После этого проверяемый документ готов к сравнению с другими документами коллекции. Если сигнатуры документов совпали – значит, они являются нечеткими дубликатами.

5. РЕЗУЛЬТАТЫ

Результатом проведенной работы стала система, позволяющая пользователям свободно индексировать электронные ресурсы научных библиотек и проверять документы на уникальность в контексте конкретной научной библиотеки. Пользователь взаимодействует с системой посредством интерфейса (Рис. 4).

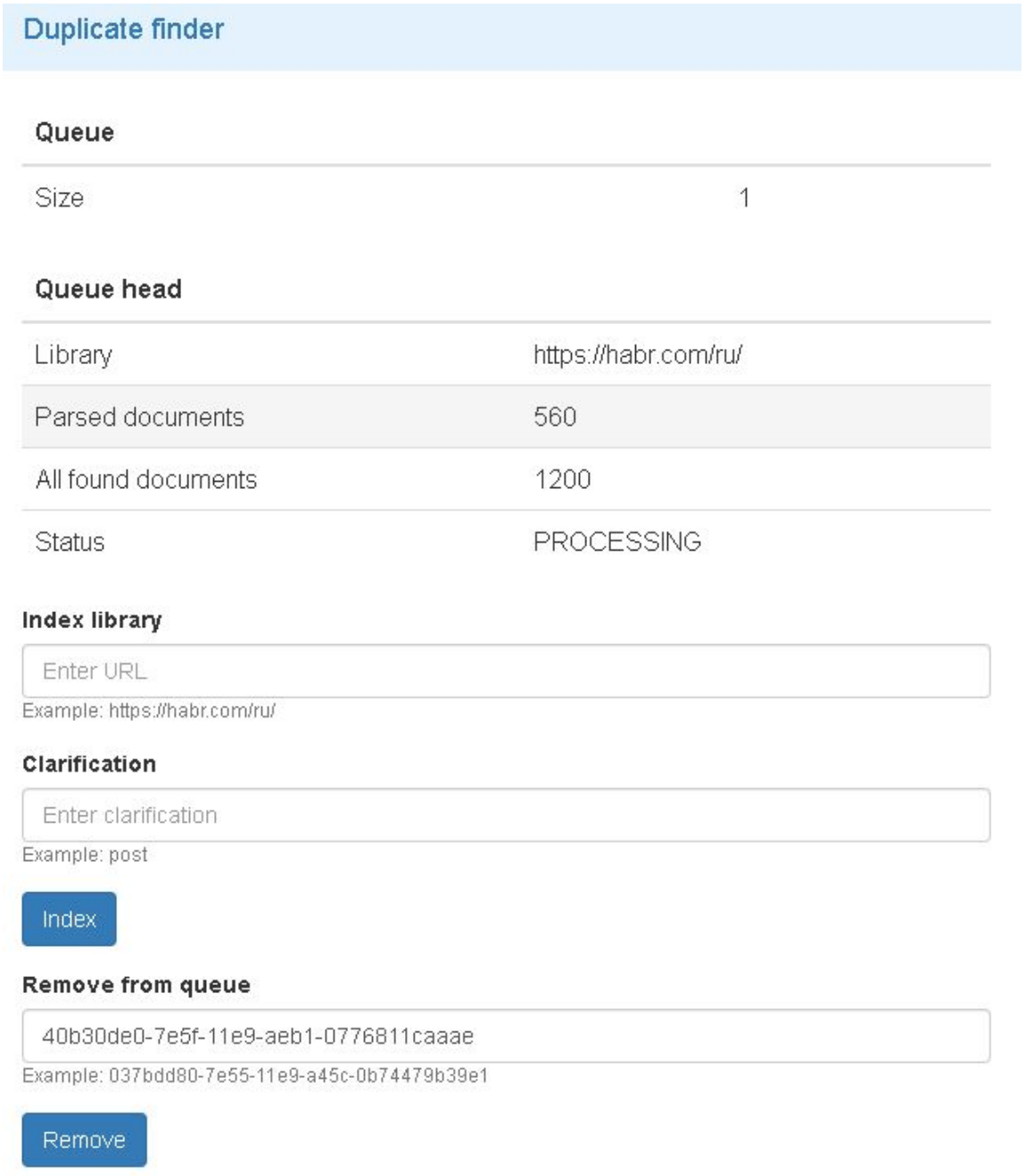

Рис. 4. Интерфейс системы для работы с очередью

Пользователь может добавлять библиотеки на индексацию, доставать библиотеку, поставленную им на индексацию из очереди, в режиме реального времени наблюдать процесс обработки электронного ресурса научной библиотеки. Процесс проверки на дубликаты осуществляется также с помощью интерфейса (Рис. 5, 6, 7).

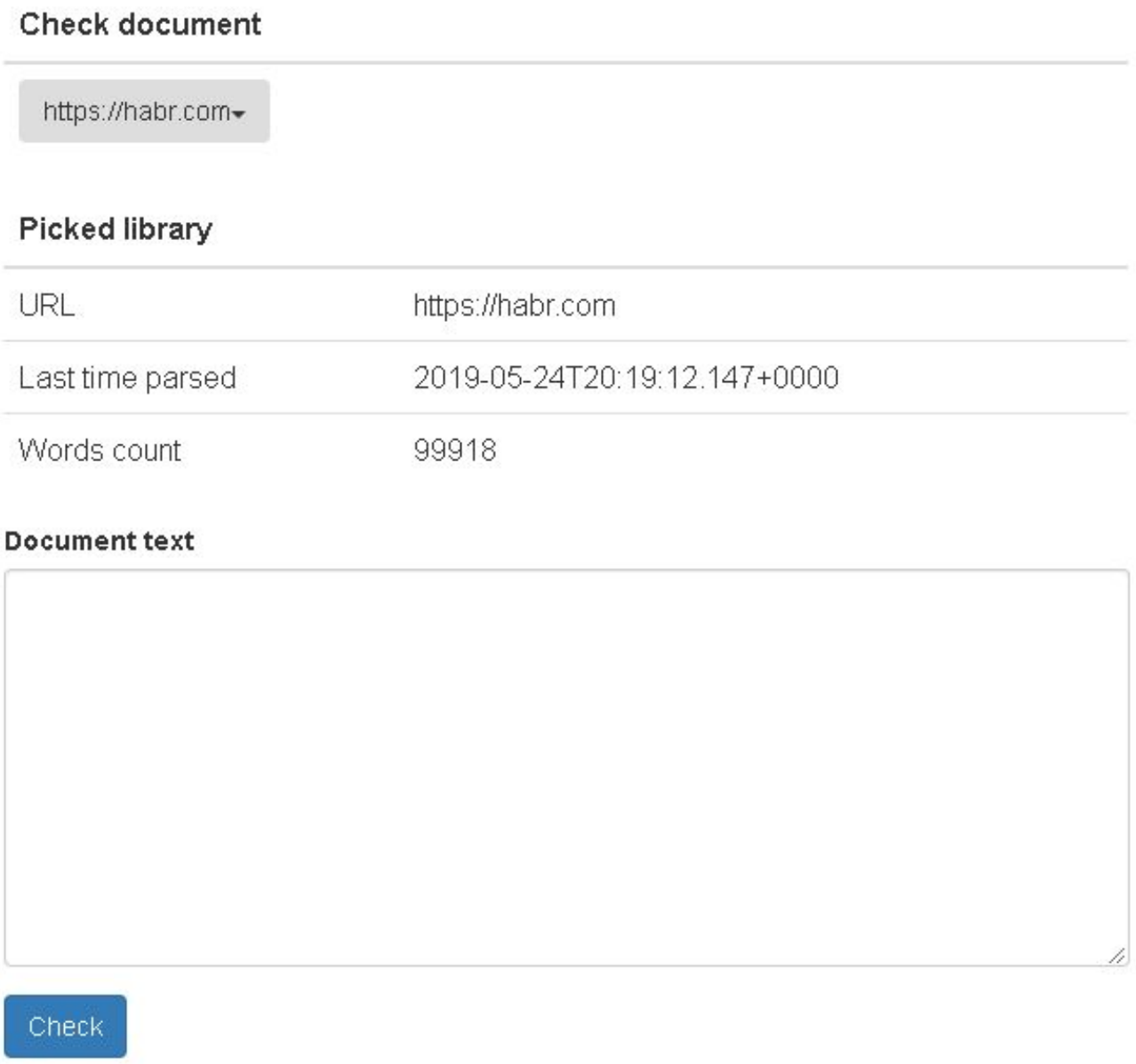

Рис. 5. Интерфейс системы для проверки на дубликаты

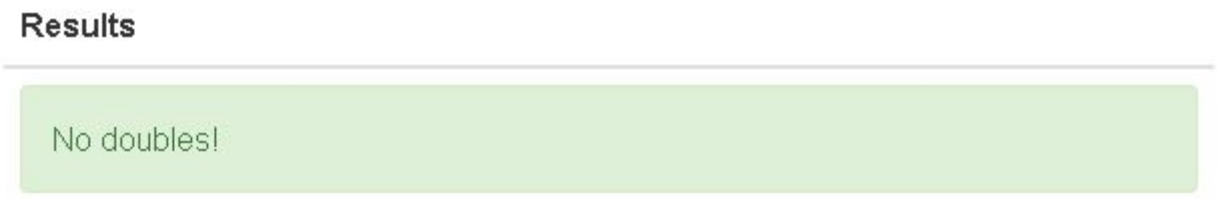

Рис. 6. Результат работы системе при отсутствии дубликатов

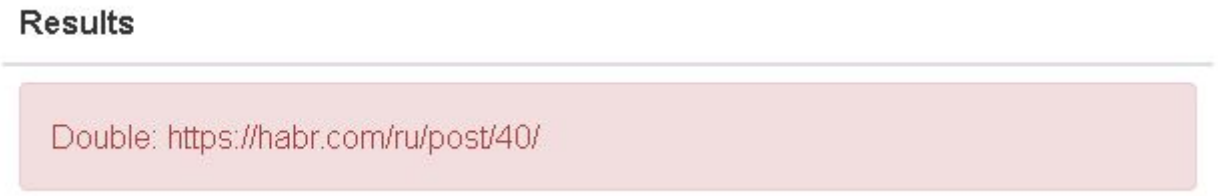

Рис. 7. Результат работы системе при наличии дубликатов

Производительность системы оценивалась на двух основных критериях:

- Скорость пересчета числового значения важности слова;
- Скорость проверки входного документа на наличие дублирования в конкретной библиотеке.

Производительность реализованного решения измерялась на тестовой среде, состоящей из процессора i5–8600 @ 3.1 GHz, RAM 8 GB.

Результаты теста производительности:

- Пересчет TF–RIDF

Количество документов в базе данных	Время (ms)
211	2096
221	1443
231	1192
252	1746
1968	12328
1998	11564
2180	15648
2529	18875

- Проверка дублирования

Количество документов в базе данных	Время (ms)
30	313
47	230
58	203
60	178
86	223
87	305
473	227
1944	206

На основании теста производительности можно заметить, что система выполняет свою основную задачу, а именно – проверку на дублирование, с оптимальной скоростью, независимо от количества статей в базе данных. Пересчет же значимости слов заметно замедляется, так как скорость работы напрямую зависит от общего количества слов в научной библиотеке. Но критичным это последствие не является, так как пересчет выполняется только один раз, по завершении индексирования библиотеки.

Итогом является приемлемое выполнение пересчета значимости слов и моментальные проверки на дубликаты.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было сделано следующее:

- Исследованы способы организации данных в цифровых научных библиотеках и определен оптимальный способ сбора данных;
- Определена оптимальная архитектура базы данных, позволяющая добавлять и обновлять данные за приемлемое время;
- Исследованы алгоритмы поиска дубликатов и выбран алгоритм, работающий с хорошими показателями как времени, так и точности;
- Реализована система поиска дубликатов для цифровых научных библиотек на основе алгоритма определения значимости слов TF–RIDF, позволяющая пользователям индексировать необходимые им научные библиотеки.

Поставленная цель – разработка системы поиска дубликатов документов для цифровых научных библиотек – была достигнута.

Данная работа размещена на gitlab – http://gititis.kpfu.ru/Romanov/duplicate_finder.

Реализованное решение может быть использовано как цифровыми научными библиотеками для проверки новых документов на наличие дублей внутри их баз данных, так и любыми другими пользователями для проверки оригинальности документов научной тематики.

В дальнейшем может быть сделано следующее:

- Поддержка различных форматов данных проверяемых документов;
- Использование какой–либо базы научных библиотек для автоматического индексирования новых библиотек.

ГЛОССАРИЙ

- API (Application programming interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой.
- Concurrent – пакет для эффективной организации многопоточности в Java.
- ConcurrentLinkedQueue – класс из библиотеки Guava, реализующий неограниченную по емкости и ориентированную на многопоточное исполнение очередь.
- CRC32 (Cyclic redundancy check 32) – алгоритм нахождения контрольной суммы, предназначенный для проверки целостности данных.
- ExecutorService – класс из пакета concurrent, позволяющий исполнять асинхронный код в одном или нескольких потоках.
- Guava – это набор библиотек с открытым исходным кодом для Java, помогающий избавиться от подобных часто встречающихся шаблонов кода.
- I–Match – Алгоритм определения нечетких дубликатов текста, основанный на лексических принципах, основной идеей которого является построение I–Match “отпечатка” документа.
- Jsoup – это библиотека для Java с открытым исходным кодом, предназначенная для анализа, извлечения и управления данными, хранящимися в документах HTML.
- Lombok – проект по добавлению дополнительной функциональности в Java с помощью изменения исходного кода перед Java компиляцией.
- Mystem–scala – библиотека для Java, содержащая в себе морфологический анализатор русского языка с поддержкой снятия морфологической неоднозначности.

- PostgreSQL – свободная объектно–реляционная система управления базами данных.
- RESTful – это общие принципы организации взаимодействия приложения/сайта с сервером посредством протокола HTTP, которые можно описать в нескольких операциях: каждый URL является ресурсом, GET возвращает описание этого ресурса, POST добавляет новый ресурс, PUT изменяет ресурс, DELETE удаляет ресурс.
- SHA1 (Secure Hash Algorithm 1) – алгоритм криптографического хеширования. Для входного сообщения произвольной длины алгоритм генерирует 160–битное (20 байт) хеш–значение.
- Spring Boot – проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web–приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.
- TF–IDF (Term frequency–inverse document frequency) – статистическая мера, используемая для оценки важности слова в контексте документа.
- Нечеткий дубликат – неполное или частичное совпадение объекта с другим объектом подобного класса.
- Стемминг – процесс нахождения основы слова для заданного исходного слова.
- Хеш–функция – функция, осуществляющая преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определенным алгоритмом.
- Шингл – это фрагмент текста длиной в несколько слов, с которым работает программа проверки уникальности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гасфилд Д. Строки, деревья и последовательности в алгоритмах [Текст] / Гасфилд Д. // СПб.: Невский диалект, 2003. – 654 с.
2. Broder A. On the resemblance and containment of documents [Text] / A. Broder. // Compression and Complexity of Sequences – 1998. – P. 21–29 .
3. Chowdhury A. Collection statistics for fast duplicate document detection [Text] / Chowdhury A., Frieder O., Grossman D., Mccabe C. // ACM Trans. Inform. Syst. – 2002. – Volume 20. – P. 171–191.
4. Church K. Poisson mixtures [Text] / K. Church, W. Gale. // Natural Language Engineering – 1995. –P 163–190.
5. Fetterly D. A large–scale study of the evolution of web pages [Text] / D. Fetterly, M. Manasse, M. Najork, J. Wiener. // In Proceedings of the twelfth international conference on World Wide Web, Budapest, Hungary, – 2003. – P. 669–678.
6. Heintze N. Scalable document fingerprinting [Text] / Heintze N. // 2nd USENIX Electronic Commerce Workshop Proceedings. – 1996. – P. 191–200.
7. Kolcz A. Improved robustness of signature-based near-replica detection via lexicon randomization [Text] / A. Kolcz, A. Chowdhury, J. Alspector // KDD '04 Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. – 2004. – P. 605–610.
8. Manber U. Finding Similar Files in a Large File System [Text] / Manber U. // WTEC'94 Proceedings of the USENIX Winter 1994 Technical Conference – 1994. –P. 2–2.
9. Postgres Pro Standart – PostgreSQL documentation [Электронный ресурс] // postgrespro.ru URL: https://postgrespro.ru/docs/postgrespro/9.5/sql–vacuum (дата обращения: 22.05.2019).
10. Zelenkov Yuri. G. Comparative analysis of near–duplicate detection methods of Web documents [Text] / Zelenkov Yuri. G., Segalovich Ilya V. // IX

All–Russian Scientific Conference “Digital Libraries: Advanced Methods and Technologies, Digital Collections” Proceedings. – 2007. – P. 166–174.