Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Разработка платформы сбора, структурирования, хранения и поиска данных, публикуемых пользователями интернета

Дипломная работа студента 545 группы

Щитинина Дмитрия Анатольевича

Научный руководитель _____к.х.н., Изъюров А.Л.

Рецензент _____Оносовский В.В.

"Допустить к защите"

заведующий кафедрой _____д.ф-м.н, проф. Терехов А.Н.

Санкт-Петербург

2011

Saint-Petersburg State University
Mathematics and Mechanics Faculty

Software Engineering Department

Development of a platform for extraction, structurization, storage and search for user generated content

Graduate paper by
Dmitry Schitinin
545 group

Scientific advisor _____ PhD A.L. Izyurov

Reviewer _____ V.V. Onossovski

"Approved by"
Head of Department _____ Professor A. N. Terekhov

Saint-Petersburg
2011

Оглавление

Глава 1

Введение

Под данными, публикуемыми пользователями интернета (в англоязычной литературе существует устоявшийся термин "User Generated Content (UGC) или "Consumer Generated Media (CGM)"), принято понимать произвольный материал, созданный и опубликованный в интернете обычными пользователями, будь то комментарий, оставленный на форуме, либо высококачественное видео, загруженное на YouTube.com, либо профиль в социальной сети.

UGC в том или ином виде существовует с момента появления интернета и по сей день. Но в последнее время, благодаря доступности высокоскоростных интернет-соединений и мощных технологий поиска, UGC стал одной из основных и самой быстрорастущей формой данных в сети. [1]

UGC коренным образом меняет то, как пользователи взаимодействуют с интернетом, а также то, как рекламодетаели привлекают аудиторию на свои ресурсы.

1.1 История

UGC являлся основным элементом peer-to-peer взаимодействия на заре цифровой эпохи. Самые ранние его формы происходят из 1980 из Uniset - мировой сети обсуждения, позволяющей пользователям оставлять комментарии или делиться опытом по тому или иному вопросу.

В конце 1990-х виден подъём оценочных сайтов ("rating sites"), предоставляющих пользователям возможность оценить тот или иной предмет по разным признакам, начиная от его физических свойств и внешнего вида, и заканчивая его описанием на профессиональном уровне. Это явление быстро распространилось по интернету, принеся вместе с собой споры по поводу влияния, которое может быть оказано на повседневную жизнь пользователей. Часто эти споры и дискуссии попадали под пристальное наблюдение публики, что еще больше

повышало популярность этих сайтов.

Другая ранняя форма UGC - это форумы - области тематических сайтов, которые позволяют пользователям общаться друг с другом по какому-либо вопросу, подпадающему под тематику сайта. Несмотря на то, что в настоящее время преобладают социальные сети, форумы продолжают быть устойчивыми и контролируемыми областями данных, публикуемыми пользователями.

1.2 Сайты с отзывами

Один из самых значимых и привлекательных типов UGC-сайтов для пользователей - это сайты с отзывами о различных товарах или услугах. Здесь потребители описывают свой опыт для того, чтобы предоставить другим пользователям больше информации для более эффективного выбора. Как правило, такие сайты разделены по категориям: товары, автомобили, туризм и т.д.

По данным исследований компании Nielson[1] потребители больше всего доверяют рекомендациям среди всех прочих видов рекламной деятельности. Результаты этих исследований приведены на рис. 1.1.

Как правило, сайты с отзывами это именно те ресурсы, где потребители могут найти такие рекомендации.

Далее будут рассмотрены некоторые популярные сайты с отзывами.

CNet

CNet основан в 1993 году и призван помогать потребителям в выборе различных электронных устройств. Редакторы сайта сами пишут отзывы и предоставляют изображения и видео, позволяющие узнать всю "правду"об устройствах от сотовых телефонов и цифровых камер до DVD-проигрывателей. Наравне с профессиональными отзывами, обычные пользователи также могут оставлять свои, которые формируют отдельный раздел сайта. Также пользователям предлагается оценить продукт по шкале от 1 до 10. Далее вычисляется средняя оценка и отображается на странице товара.

[1]Nielson - ведущая мировая информационная и аналитическая компания, занимается изучением рынка. Ключевые направления "Что смотрят люди? "Что покупают люди?". www.nielsen.com

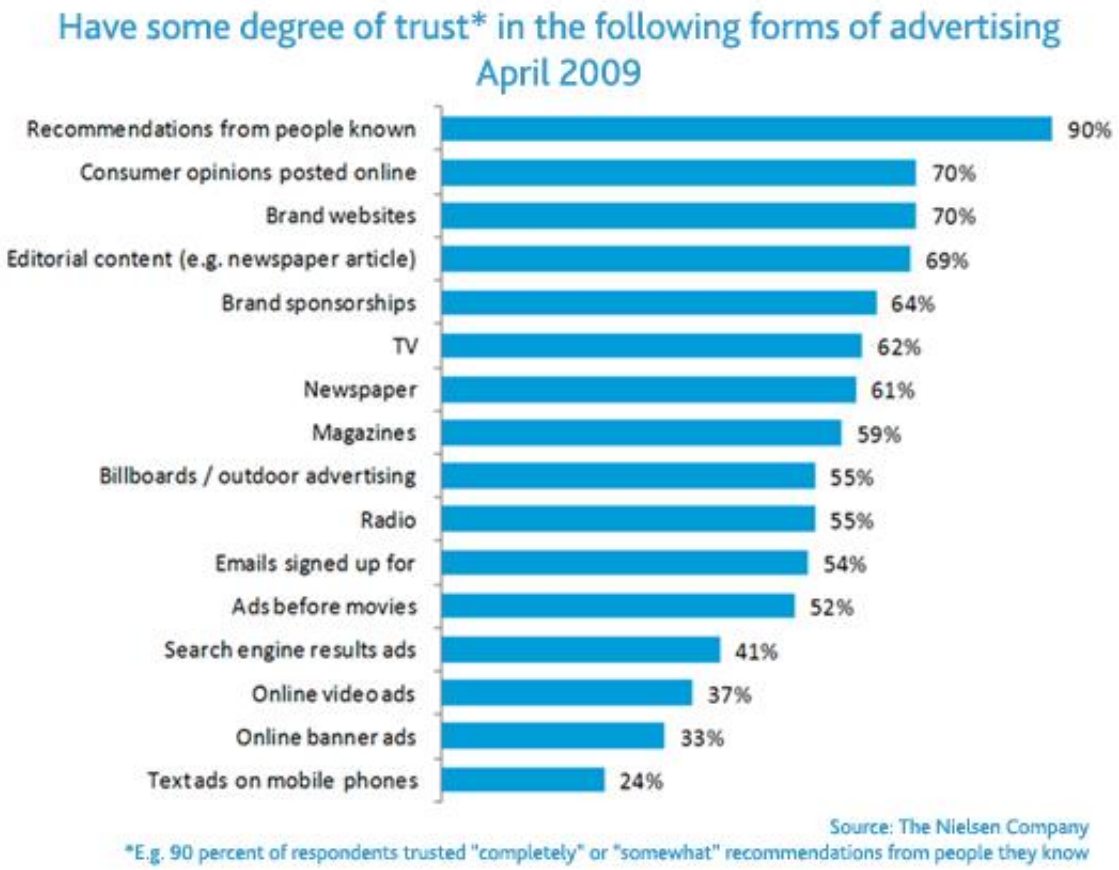

Рис. 1.1: Уровень доверия к различным видам рекламной деятельности (по данным Nielson)

Edmunds.com

Данный ресурс аналогичен CNet, но только для автомобилей. Редакторы сайта публикуют свои отзывы, фото и характеристики. Также существует выделенная область сайта, где пользователи делятся своим опытом владения автомобилем. Все содержимое, публикуемое пользователями на Edmunds.com, проходит модерацию, поэтому оскорбительные и вызывающее материалы сразу удаляются.

Яндекс.Маркет

Яндекс.Маркет — это система выбора разнообразных товаров и места для их покупки. К услугам пользователей подробные описания характеристик товаров, подбор товара по параметрам, сравнение моделей и цен, отзывы покупателей о товарах и магазинах. Все отзывы также проходят модерацию и материалы, не проходящие проверки на качество, не публикуются.

1.3 Постановка задачи

Как было показано, отзывы являются крайне полезным материалом при выборе товара или услуги. В настоящее время, благодаря широкому распространению интернета, появляется множество сайтов с отзывами, различающихся как по объему публикуемых данных, так и по их качеству, не говоря уже о способах представления. Как правило, все такие сайты (даже самые популярные, описанные в первой главе) позволяют лишь опубликовать отзыв на некоторый объект, соответственно, способны осуществлять поиск только по "своим" отзывам. Все эти факторы создают различного рода неудобства для пользователей, которые стремятся найти отзывы на конкретный товар или услугу. Поисковые системы позволяют найти сайты, и даже конкретные страницы с отзывами на интересующий объект, но пользователю, тем не менее, придется потратить время на то, чтобы понять качество источника в целом, можно ли ему доверять, также разобраться с навигацией на ресурсе, возможно, провести какую-либо аналитику. Также не все сайты предоставляют возможность фильтрации отзывов по различным параметрам (например, по рейтингу или эмоциональной окраске - на практике оказывается, что отрицательные отзывы более информативны).

Задача, поставленная перед автором дипломной работы и сотрудником компании "Яндекс" состоит в разработке платформы для сбора отзывов с различных источников, их хранения, обработки, осуществления поиска по собранным данным, а также внедрения плат-

формы в тематические сервисы Яндекса.

Наличие такой платформы позволит осуществлять поиск по отзывам на интересующий объект среди данных многих источников, предварительно обработанных и отфильтрованных по качеству. Как следствие, результаты поиска с разных источников будут представлены одинаково, будут "сквозными" по всем источникам, с возможностью различного ранжирования и фильтрации. Хранение собранных данных позволит проводить их различный анализ, например, извлечение фактов, что является для конечного пользователя "вкусной" информацией.

Глава 2

Обзор существующих решений

Поскольку платформа состоит из частей, осуществляющих сбор данных, их хранение, обработку и поиск, существующие подходы к реализации каждой из них будут рассмотрены по-отдельности.

2.1 Сбор данных

Экстракция данных (information extraction) - этап информационного поиска (information retrieval), задачей которого является извлечение структурированной информации из неструктурированных или полуструктурированных электронных документов.

2.1.1 Системы извлечения данных из веба

Задача экстракции данных по отношению к интернету решается системами извлечения данных из веба (Web Data Extraction Systems). Это программное обеспечение, которое регулярно автоматически извлекает информацию с веб-страниц, совершая необходимые изменения извлеченных данных и сохраняя результат в базу данных или передавая его в соответствующее приложение.

Функции систем извлечения данных

При рассмотрении систем извлечения данных из веба принято выделять пять основных функций:

- Взаимодействие с вебом (web interatcion)
  Функция отвечает за навигацию по, как правило, заранее определенному набору целевых веб-страниц.

- Поддержка создания "оберток"и их выполнение

  "Оберткой"(wrapper) называется программа, которая находит требуемые данные на целевых страницах, извлекает эти данные, при этом преобразуя их в структурированный формат.

- Управление задачами (scheduling)

  Отвечает за регулярные запуски "оберток".

- Преобразование данных

  Сюда входят фильтрация, трансформация, очистка извлеченных данных. А также интеграция данных, извлеченных с различных источников. Результирующие данные представлены в некотором фиксированном формате, как правило, это XML.

- Транспорт сформированных структурированных данных к внешним приложениям, таким как СУБД, хранилища данных, информационные системы, системы принятия решений, сервера электронной почты, SMS-сервера и т.д.

История

Первыми предшественниками систем извлечения данных из веба могут считаться "экранные скребки"(screen scrapers), предназначенные для извлечения данных из форматированного вывода приложений, работающих на мейнфрэймах.

Другой связанный предшественник - это ETL (Extract-Transform-Load) процессы, извлекающие данные из различных бизнес-процессов и сохраняющие результаты в базу данных или склад данных (data warehouse).

Естественным продолжением "экранных скребков"стали "веб-скребки"(web scrapes). Как правило, они не могли формировать хорошо структурированный результат и ограничивались в основном цепочками текстов или сниппетами. Тем более, эти системы не умели интегрировать результаты из разных источников, осуществлять способы экстракции, которые оставались бы устойчивыми к небольшим изменениям разметки, преобразовывать результат в требуемый формат и доставлять качественные данные к потребителям. Указанные недостатки вызывали интерес со стороны многих академических и коммерческих проектов, в результате чего был создан ряд как свободно распространяемых, так и проприетарных продуктов.

Способы создания оберток

По степени вовлеченности пользователя различают следующие подходы для создания оберток:

- Ручное программирование

  Пользователь пишет конкретные обертки для конкретных страниц, но при этом не имеет возможности что-либо обобщать и переиспользовать

- Индукция

  Пользователь предоставляет примеры и контрпримеры шаблонов данных, которые требуется извлечь, система на их основе производит требуемые обертки (как правило, для этого используется машинное обучение).

- Полуавтоматическая интерактивная генерация

  При этом подходе создатель оберток не только приводит примеры данных системе, но также сопровождает генерацию обертки через интерактивное взаимодействие с системой, включающее обобщение, корректировку, тестирование.

- Семантическая разметка со стороны ресурса

  К таким разметкам относится, например, микроформатная разметка.

Архитектура систем извлечения данных

В своей работе [3] авторы приводят пример типичной архитектуры полнофункциональной полуавтоматической интерактивной системы извлечения данных.

Система состоит из различных компонент, призванных связать три внешних сущности:

- **Веб**, в котором находятся ресурсы и страницы с нужной информацией
- **Целевое приложение**, которому будут переданы извлеченные обработанные данные
- **Пользователь**, интерактивно создающий обертку

Сама система состоит из следующих компонент:

- **Wrapper generator** (генератор оберток)

  Компонент призван помогать пользователю создавать обертку, предоставляя различные вспомогательные данные (браузер, модель страницы и т.п.), а также интерфейс для интерактивного взаимодействия. При поддержке пользователя генератор создает обертку.

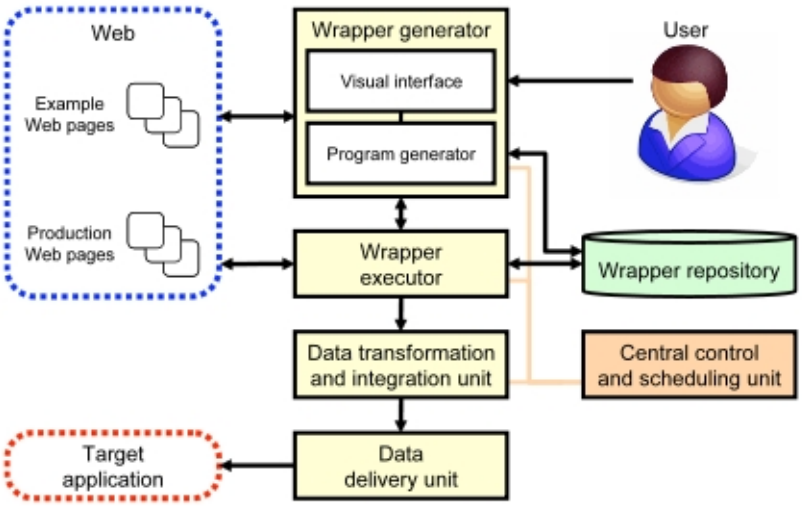

Рис. 2.1: Архитектура систем извлечения данных из веба

- **Wrapper executor** (исполнитель оберток)

  Компонент отвечает за развертывание созданных оберток, навигацию по вебу, определение нужных страниц, применение обертки и создание структурированного результата.

- **Wrapper repository** (хранилище оберток)

  Компонент отвечает за хранение созданных оберток совместно с некоторыми метаданными и предоставление доступа к ним.

- **Data transformation and integration** (преобразование и объединение данных) Компонент отвечает за объединение результатов работы различных оберток в один целостный результат. Также здесь осуществляется различные преобразования данных и фильтрация.

- **Data delivery unit** (модуль загрузки данных)

  Компонент отвечает за выбор нужного канала выходных данных, таких как электронная почта, СУБД, RSS, FTP и т.п.

- **Central control and scheduling unit** (модуль управления и планирования)

  Отвечает за управление запусками оберток. В некоторых случаях может содержать сложную логику таких запусков.

2.1.2 Микроформаты

Микроформаты являются одним из способов семантической разметки информации о различных сущностях (персоны, события, места, отзывы, рецепты) при помощи стандартных элементов HTML-разметки, благодаря чему системы, обрабатывающие такие страницы, способны извлекать структурированную информацию, следуя определенным соглашениям. В то же время, микроформаты это определенный набор соглашений и правил, описывающих то, как именно следует размечать ту или иную информацию. Множество различных точек зрения на то, что есть микроформат приведены в `http://microformats.org/wiki/what-are-microformats`.

К достоинствам микроформатов можно отнести простоту их использования для разметки и нетребовательность к дополнительным инструментам и языкам, кроме разметки страницы. Поддержка микроформатов крупными поисковыми системами стимулирует вебмастеров использовать этот подход. Также формат разметки предполагает различные расширения и возможность добавления новых полей.

Недостатком микроформатов является отсутствие валидации разметки. Это влечет ошибочные данные в результатах экстракции, а также наличие в них различного рода "мусора" что требует дополнительной очистки и фильтрации извлеченной информации.

2.1.3 Приложения и сервисы

Dapper - `http://open.dapper.net/dapp-factory.jsp`

Dapper - веб-сервис по создание оберток. Обертки создаются на стороне сервера с использованием машинного обучения (пользователь приводит примеры и контрпримеры данных). Имеет хранилище оберток, которые можно переиспользовать, поддерживается сообществом. Обрабатывает страницы, доступные без "глубокой навигации" (заполнение форм, ввод паролей и т.п. не производится).

Denodo - `http://www.denodo.com`

Denodo - коммерческая платформа для создания и исполнения скриптов навигации и оберток. Платформа предлагает графические средства для создания и отладки оберток, обработку различных событий DOM-модели страниц, которые могут обрабатываться при навигации.

KapowTech - `http://kapowsoftware.com/solutions/data-collection/web-data-extraction.php`

Kapow RoboSuite - среда разработки оберток, написанная на Java. Включает в себя собственный браузер, графический интерфейс, средства отображения данных в реляционные СУБД. Имеет API к различным языкам программирования.

2.2 Хранение данных

2.2.1 Полуструктурированные данные

Под **полуструктурированными данными** (semistructured data) принято понимать данные, не имеющие постоянной четко определенной структуры, способные динамически изменять свою структуру и состав. Под полуструктурированными данными также понимают данные, имеющие некоторую структуру, но она по каким-либо причинам неизвестна потребителю этих данных.

Полуструктурированные данные удобно представлять в виде ориентированного графа, каждая вершина которого является элементом данных, а каждая дуга имеет метку. Листья графа содержат данные (атомарные объекты - числа, строки, изображения, аудио, видео), а внутренние вершины являются контейнерами (составные объекты), собирающими воедино несколько объектов. Дуги, выходящие из внутренней вершины, указывают на содержащиеся в ней объекты.

2.2.2 Подходы к хранению полуструктурированных данных

В общем случае, подходы к хранению полуструктирированных данных можно поделить на три категории:

- **Плоские потоки** (flat streams)

  При этом подходе графы сериализуются в потоки байтов (например, в соответствии с языком разметки). В случае больших объёмов получающихся потоков используется механизм распределения байтовых потоков на диск. К таким механизмам относится как сама файловая система, так и механизм управление BLOB-данными в СУБД.

  Подход хорош в случае сохранения и получения документов целиком или больших непрерывных частей документов. Для доступа к элементам документа требуется его разбор, что сильно замедляет операции модификации.

- **Метамоделирование** (metamodeling)

  При этом подходе для хранения графов, представляющих структурированные данные, применяется их моделирование и хранение в терминах СУБД и ее моделей данных (будь то файловая система или реляционная СУБД)

  В этом случае доступ к отдельным элементам данных быстрее, чем в предыдущем методе. Но с другой стороны, получение всего документа или его частей для представления в текстовом виде медленнее, чем при использовании плоских потоков. Построение одного документа может потребовать извлечения сотен и тысяч объектов из базы данных. В общем случае, такой подход вводит дополнительный слой в СУБД между логическим и физическим уровнем, замедляя выполнение запросов.

- **Смешанный подход**

  Подход является попыткой объединить предыдущие два.

  - **Избыточный**

    Для получения системы, совмещающей плюсы обоих подходов, данные хранятся в двух независимых хранилищах, одно из которых подход плоских потоков, другое - метамоделирования. Это позволяет быстро получать документы целиком, но обновления все же медленны, к тому же добавляются накладные расходы на поддержку целостности и связанности данных между хранилищами.

  - **Гибридный**

    При этом подходе определенный уровень данных выбирается в качестве порога. Структуры, выше выбранного порога, хранятся в структурных объекта базы данных. Структуры, ниже порога, хранятся как "плоские" объекты базы данных.

2.2.3 XML

Стандартом де-факто для представления и передачи структурированных данных стал XML (eXtensible Markup Language), который является одним из способов записи ориентированного графа, соответствующего полуструктурированным данным.

Благодаря широкому распространению XML, предложено множество различных подходов для хранения данных, представленных XML, а также их поиска и редактирования. Практически каждому подходу соответствует реализация в открытых или коммерческих системах.

2.2.4 XML-хранилища

Благодаря широкому применению XML, многим приложениям требуется обрабатывать или управлять XML-документами, также многие приложения хранят и получают данные из больших документов или больших коллекций документов. Как формат обмена данными, XML может быть сериализован в поток и сохранен, но такой способ крайне неэффективен при обработке запросов и обновлениях. Объемы работы по обработке XML-данных могут сильно различаться от приложения к приложению: запросы или обновления могут возвращать или изменять как несколько, так и большие количества узлов. Также может потребоваться обход большой части XML-дерева. Все это приводит к тому, что для высокой производительности современных приложений требуется масштабируемые XML-хранилища с богатыми возможностями.

2.2.5 Подходы к хранение XML-данных

Существует несколько подходов для хранения коллекций XML-документов, каждый из которых по-разному преодолевает сложности, вызванные разнообразием структуры XML-документов и рабочими нагрузками. Далее приводятся основные идеи подходов, подробности которых можно узнать в статье [2]. Поскольку XML является одним из способов представления полуструктурированных данных, к XML-хранилищам применима общая классификация хранилищ полуструктурированных данных. Но в то же время их можно классифицировать по наличию или отсутствию фиксированной структуры хранимых документов:

- Instance Driven Storage

  В этом подходе способ хранения XML-данных определяется структурой конкретного документа, при этом не учитываются типы узлов, определенные схемой документа (если таковая задана). Такой подход сильно упрощает задачу хранения данных в случае, когда документы не имеют регулярной структуры или когда приложение часто обращается к отдельным элементам. Хранить узлы и ребра можно как в одной или нескольких таблицах реляционной базы данных, так и реализовать их хранение напрямую (на файловой системе).

- Schema Driven Storage

  В случаях, когда известна структура XML-документа (DTD или XML-схема), можно эффективно использовать эту информацию для построение XML-хранилища. Общая идея заключается в том, что однотипные узлы (в соответствии со схемой) отобража-

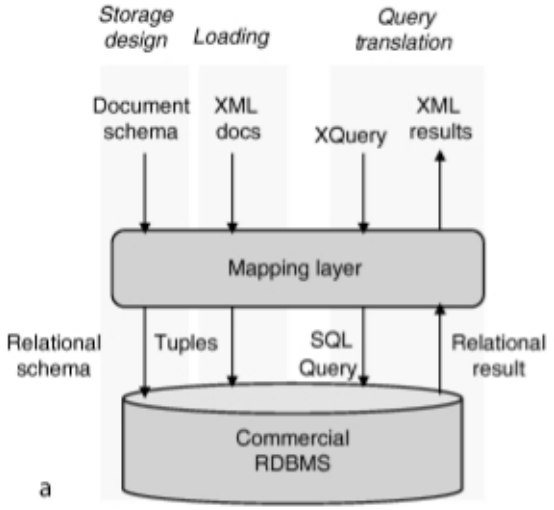

Рис. 2.2: Реляционное хранилище XML

ются единообразно (например, в таблицы реляционной базы данных). Как правило, информация из схемы используется для табличного хранения коллекций документов, в частности, может использоваться механизмом реляционного хранилища (relational storage engine).

На практике широко распространены реляционные хранилища XML-документов (Relational Storage for XML Documents). Применение этого подхода позволяет эффективно использовать реляционные СУБД для хранения XML. Для работы подход использует метамоделирование (Metamodeling), применяя знания о структуре хранимых документов (Schema Driven). На рисунке 2.2 приведены основные элементы реляционного хранилища XML, а также выполняемые ими задачи.

Сначала XML-схема отображается в подходящую реляционную схему (Storage Design). Получившаяся реляционная схема должна быть оптимизирована на физическом уровне (правильный выбор структур хранения и создание индексов) с учетом различных характеристик предполагаемых запросов и обновлений, а также рабочей нагрузки. При сохранении XML-документов они сначала дробятся на части, затем части загружаются в плоские таблицы (Data Loading). Поступающие в систему XML-запрос переводится в соответствующий реляционный запрос, передаваемый в реляционную систему, и полученные результаты переводятся обратно в XML (Query Translation).

2.2.6 Программные продукты и проекты

Наиболее известны следующие проекты для хранения полуструктурированных данных и XML.

LORE, `http://infolab.stanford.edu/lore/`

LORE (Ligthweigh Object Repository) - система хранения полуструктурированных данных, разработанная в Стэндфорском университете. В своей основе она использует модель данных OEM (Object Exchange Model) - самоописывающуюся модель с отношением вложения и уникальными идентификаторами. Данные в этой модели представляются в виде ориентированного графа. Вершины графа - объекты, имеющие уникальные идентификаторы. Объект составной, если он имеет исходящие дуги. Дуги имеют метки, описывающие отношения "объект-подобъект". Вершина, не имеющая исходящих дуг - атомарный объект. Помимо идентификатора он имеет тип (строка, число и т.п.) и значение, представляющее данные. LORE не является полноценной СУБД, оно не имеет транзакционности, управления многозадачностью и возможностей восстановления данных. LORE разрабатывалось в конце 90х годов, разработка закончена в 2000 году. На сегодняшний день уже не поддерживается. Подробнее об это системе рассказывается в [8]

BaseX - `http://basex.org`

BaseX - система хранения XML, нативная (работает поверх файловой системы). Имеет поддержку визуализации документов и удобный интерфейс редактирования. Имеет поддержку XPath/XQuery. Информацию о рабочих нагрузках авторами не публикуется. Распространяется по BSD-лицензии.

eXist - `http://www.exist-db.org`

eXist - система хранения XML с открытым исходным кодом, нативная. Для манипуляций с данными используются XPath и XQuery. Имеет транзакционность и возможность восстановления после сбоев. Может быть использован как обычный сервер базы данных, так и как подключаемая java-библиотека для локального хранения и обработки XML. Максимальное число хранимых документов ограничивается 2^{32}. Ограничения на размер одного документа зависят от используемой файловой системы. По сути, каждый документ хранится в отдельном файле на ФС. Детали реализации можно найти в [7]

Sedna - `http://modis.ispras.ru/sedna`

Sedna - система хранения XML с открытым исходным кодом (распространяется по лицензии Apach License 2.0), написанная на C/C++. Имеет поддержку транзакционности, механизмы разграничения доступа, возможности восстановления данных. Для манипуляции данными используются XPath и XQuery, имеется возможность полнотекстового поиска. Работает как выделенный сервер.

KDepot

Проприетарная разработка компании "Яндекс" для хранения полуструктурированных данных. В основе использует модель, схожую с моделью OEM, используемую в LORE. Имеет реализацию как при помощи метамодели (отображения в различные реляционные базы данных, Oracle и MySQL), так и нативную (хранение на файловой системе).

2.3 Обработка данных

2.3.1 Нахождение нечетких дубликатов

Одним из важных и наиболее трудных видов обработки данных, собранных из различных источников, является обнаружение нечетких дубликатов. Актуальность этой проблемы определяется множеством мест в системе, в которых необходимо учитывать дублирование данных. Как правило, это повышение качества поискового индекса за счет удаления избыточной информации, установление нарушений авторских прав, улучшение результатов поиска (пользователю не показывается два почти одинаковых результата) и многое другое [10]. Основную сложность в задаче нахождения нечетких дубликатов составляет объемы данных, в которых эти дубликаты требуется определять. Как правило, это коллекции от сотен тысяч до миллиардов документов.

Далее будут рассмотрены наиболее известные алгоритмы нахождения нечетких дубликатов. Все алгоритмы нацелены на уменьшение вычислительной сложности за счет применения различных эвристик (хеширование некоторого набора слов документа, использование дактилограмм и др.). Алгоритмы могут быть поделены на две категории:

- Локальные - для вычисления сигнатуры документа требуется только сам документ
- Глобальные - для вычисления сигнатуры нужны знания обо всей коллекции

Сравнение контрольных сумм от целых документов

Тривиальный случай - определение точных дубликатов документов. От документа вычисляется контрольная сумма (например, MD5) и два документа считаются точными дубликатами, если у них она совпадает. Разумеется, о хорошей полноте и высокой точности этого алгоритма говорить не приходится.

Синтаксические методы

В конце 90-х годов A. Broder [4] был предложен следующий метод. Документ разбивается на цепочки фиксированной длины, состоящие из соседних слов. Такие последовательности называются "шинглами" (shingles). Идея синтаксических методов в том, что два документа считаются похожими, если их множества шинглов сильно пересекаются. Поскольку количество шинглов документа есть $O(n)$, где n - количество слов в документе, то предпринимаются разнообразные подходы для уменьшения этого значения (как следствие, памяти, потребляемой алгоритмом).

При одном подходе, в рассмотрении участвовали только шинглы, хеши которых делились на некоторое число. Основной недостаток - зависимость от длины документа. При другом подходе оставлялись только заданное число шинглов, имеющих экстремальные значения хешей.

Следующим этапом развития синтаксических методов являются исследования D. Fetterly [6]. Для каждого шингла вычислялось n функций, затем составлялся n-мерный вектор v, такой, что $v[i] = \min_{s \in Shingles} f_i(s)$ (минимальное значение i-ой функции на всем наборе шинглов документа). Вектор назывался вектором минимальных значений. Затем бралось число m, такое, что n делится нацело на m, и координаты вектора разбивались на m групп, значения в каждой из которых упорядочивались и считалась контрольная сумма. В результате получалось $k = \frac{n}{m}$ значений, которые назывались *супершинглами*. Два документа считаются нечеткими дубликатами, если у них совпадало хотя бы l супершинглов. Плюсы такого подхода в том, что любой документ, независимо от длины, представляется фиксированным числом значений.

Лексические подходы

A. Chowdhury [5] был предложен подход, идея которого заключается в вычислении сигнатуры *I-Match* для представления содержания документа. Сначала для заданной коллекции документов вычисляется словарь L, содержащий слова со средним значением IDF (Inverse

Document Frequency, $IDF_t = \log \frac{N}{df_t}$, где N - число документов в коллекции, df_t - число документов, содержащих терм t), т.к. слова с наибольшим значением IDF, как правило - это служебные части речи, а с наименьшим - опечатки и прочие ошибки. Далее, для каждого документа считается множество его слов U и определяется пересечение со словарем L. Если пересечение больше заданного порога, то слова, входящие в пересечение, упорядочиваются и от них считается хеш-функция SHA-1. Полученное значение и есть *I-Match* сигнатура. Документы считаются дубликатами, если у них совпадает *I-Match* сигнатура.

Сравнение подходов

В свой работе Hui Yang и Jamie Callan [9] проводят сравнительный анализ методов поиска нечетких дубликатов. Свои тесты они проводили на следующих наборах данных: EPA - 500000 электронных писем, DOT - 40000 электронных писем и pdf-документов. Далее из этих наборов данных были выделены небольшие подмножества для их оценки асессорами. Авторами были получены следующие результаты:

Категория	Алгоритм	Средняя точность		Средняя полнота	
		EPA	DOT	EPA	DOT
Точный дубль	Шинглы	0.97	0.93	0.91	0.90
	I-Match	0.98	0.97	0.75	0.60
Небольшое изменение	Шинглы	0.90	0.88	0.79	0.80
	I-Match	0.90	0.90	0.78	0.80
Добавление блока	Шинглы	0.73	0.30	0.32	0.20
	I-Match	0.78	0.30	0.42	0.30
Удаление блока	Шинглы	0.72	0.72	0.30	0.35
	I-Match	0.78	0.70	0.40	0.35
Перемещение блоков	Шинглы	0.67	0.50	1.00	0.90
	I-Match	0.83	0.56	1.00	0.87

Как видно, алгоритмы по-своему чувствительны к тем или иным преобразованиями исходных документов. Поэтому выбор алгоритма будет зависеть от конкретной задачи.

С точки зрения реализации локальные и глобальные алгоритмы также различаются. При применении локального алгоритма достаточно одного прохода по данным, при применении глобального требуется два прохода (первый - сбор статистики по всей коллекции документов, второй - само нахождение нечетких дубликатов), либо требуется хранить эту статистику и

корректировать ее при изменениях коллекции.

2.4 Поиск данных

В общем случае, поиск по собранным и сохраненным данным можно осуществлять двумя способами:

- С использованием механизмов поиска, реализованных в системах хранения
- С использованием внешних индексов

Возможность использования механизмов поиска систем хранения может сильно меняться от одной системы к другой (одни системы поддерживают поиск, другие нет, одни хорошо обрабатывают один тип запросов, другие - другой), что объясняется различными подходами и реализациями этих систем.

При использовании внешних индексов следует следует обращать внимание на следующие характеристики:

- Скорость индексирования

  Как быстро индексатор обрабатывает документы и заносит их в индекс.

- Скорость переиндексации

  Документы в хранилище изменяются или добавляются новые, поэтому индекс приходится обновлять. Индексатор может иметь поддержку инкрементного индексирования (добавление в индекс только новых документов). При отсутствии такой возможности потребуется полное перестроение индекса.

- Наличие API

  При разработке приложения, работающего с поисковым модулем, важно, чтобы модуль имел API к языку или платформе, на которых разрабатывается приложение.

- Ограничения на размер индекса и скорость поиска

- Работа с различными языками

  Для качественного поиска необходима работа с особенностями языка. Как правило, все известные поисковые механизмы поддерживают английский язык, но, например, для русского языка необходимо искользовать различные средства, обеспечивающие морфологию.

- Поддержка дополнительных типов полей в документах

  Кроме самого текста, который индексируется и по которому производится поиск, иногда бывает важно хранить некоторые поля, содержащие дополнительную информацию о документе, а также иметь возможность гибко настраивать индексируемость этих полей.

- Наличие механизмов ранжирования и сортировки

  Иногда требуется возможность расширения поискового модуля и специфическая реализация функций ранжирования и сортировки.

Далее будет приведен обзор наиболее известных поисковых решений. Все они основаны на обратном индексе.

Sphinx - http://sphinxsearch.com

Sphinx является кроссплатформенным программным продуктом, написанным на C++. Существует реализация как в виде отдельного сервера, так и механизма хранения для MySQL. Имеет возможность работы с дельта-индексами (инкрементальная индексация). Обрабатывает текст или XML-документы в собственном формате. Поисковый документ может содержать неограниченное количество дополнительных полей. Скорость работы очень высока: индексация порядка 10Мб/с, поиск порядка 100 мс в индексе 4 Гб. Поддерживает индексы объемом сотни гигабайт и сотни миллионов документов. Распространяется по GPL2 лицензии.

Lucene - http://lucene.apache.org

Lucene - кроссплатформенный программный продукт, написанный на Java. Распространяется как выделенный сервер, так и в виде встраиваемой библиотеки. Поддерживает инкрементальную индексацию (можно производить параллельно с поиском). При использовании Lucene как библиотеки, появляется возможность обрабатывать произвольные данные, формирование которых возлагается на плечи разработчика. Поисковый документ также может содержать неограниченное количество дополнительных полей. Продукт с открытым исходным кодом, распространяется по Apache License 2.0.

Яндекс.Сервер - http://api.yandex.ru/server

Яндекс.Сервер - кроссплатформенный программный продукт, написанный на C++. Распространяется как выделенный сервер. Поддерживает инкрементальную индексацию, возможность поиска по различным коллекциям и объединения результатов. Поддерживает произвольное число дополнительных полей. Имеет хорошую поддержку русского языка. Умеет

индексировать данные из разных источников (веб, файловая система, содержимое баз данных через ODBC) и в разных форматах.

Глава 3

Описание платформы

Разрабатываемая платформа предназначалась для сбора данных, публикуемых пользователями интернета, их структурирования, хранения и осуществления по ним поиска. При этом внимание фокусировалось на отзывах на различные товары и услуги, как на одном из самых полезных типов UGC.

Перед платформой ставились следующие требования:

- Сбор отзывов различных типов с различных веб-сайтов с возможностью дешевого подключения нового источника
- Возможность публикации отзывов от пользователей сервисов (API для оставления отзыва)
- Связывание отзыва с объектом, на который отзыв оставлен
- API параметрического поиска отзывов
- Простота использования и внедрения в различные контент-сервисы Яндекса
- Максимально быстрое с момента публикации попадание отзыва в индекс

3.1 Мотивация выбора компонентов

Поскольку разрабатываемая платформа предназначалась к промышленному использованию в крупной компании, это повлекло некоторые ограничения на выбор компонентов для различных частей системы. Например, совершенно недопустимо использование для решения конкретных бизнес-задач закрытых коммерческих приложений и сторонних сервисов, то есть все то, что не может быть модифицировано по тем или иным причинам. Также следует с

осторожностью относится к не применявшимся ранее открытым продуктам, какими бы привлекательными они не казались, отдавая предпочтение проверенным, зарекомендовавшим себя решениям. Лишь после тщательного тестирования на требуемых нагрузках и объемах данных, новые средства можно вводить в эксплуатацию.

Сбор данных

В качестве источников данных выбирались ресурсы, имеющие микроформатную разметку hReview (`http://microformats.org/wiki/hreview`). Такой выбор обусловлен, во-первых, наблюдением, что, как правило, разметку имеют данные более высокого качества, а во-вторых, это быстрый и простой способ определить наличие на странице требуемого содержания и извлечь его. Но нет правил без исключений: существуют ресурсы с данными очень высокого качества, но не имеющие микроформатной разметки. Таких источников совсем немного, поэтому к ним обоснованно применять системы извлечения данных с обертками, созданными ручным или полуавтоматическим способами. Для сбора данных применялись исключительно внутренние продукты и технологии компании и автором дипломной работы ничего нового предложено не было.

Хранение данных

Для хранения собранных полуструктурированных данных была выбрана проприетарная технология KDepot. Технология имеет реализацию как метамоделированием поверх реляционных баз данных (отображая полуструктурированные данные в некоторую реляционную схему, при этом используя гибридный подход - данные, выше определенного уровня отображаются в струткурированные элементы базы данных, данные, ниже этого уровня - в плоские потоки - BLOB/CLOB), так и нативную (хранение данных на файловой системе). На такой выбор повлияло наличие API на Java (на которой велась вся разработка), внутренняя поддержка проекта, применение его в некоторых других проектах, уверенность в его стабильности и четкое понимание ограничений использования. При выборе технологии обращалось внимание на eXist и BaseX, но требуются более глубокий их анализ перед промышленным применением.

Поиск

При выборе поискового решения основное внимание уделялось следующим возможностям:

- Инкрементальная индексация

  Обеспечивает быстроту попадания свежих данных в поисковый индекс.

- Наличие дополнительных полей в поисковом документе

  Возможность задавать сложные запросы ("вернуть все отзывы на заданный объект с заданным рейтингом").

- Удобный API к языку, на котором ведется разработка

  Повышает скорость разработки, упрощает поддержку и добавление нового функционала.

Поскольку предполагается параметрический поиск данных, то возможности полнотекстового поиска рассматриваемых решений во внимание не принимались. Инкрементальную индексацию и поддержку дополнительных полей в документах имеют все рассмотренные поисковые решения. Также все они являются проверенными во многих проектах и не вызывают сомнений в надежности и производительности. Но удобный API к языку, на котором ведется разработка (Java) - только Lucene, распространяемый как библиотека к нему. В нашем случае это преимущество стало ключевым и выбор пал на Lucene.

Обработка данных

Одним из самых сложных этапов обработки данных является поиск и устранение нечетких дубликатов. В случае с отзывами это проблема стоит особенно остро, так как дубли встречаются довольно часто. Во-первых, это недобросовестные сайты, которые в буквальном смысле воруют данные с других источников, во-вторых, сами пользователи, стремясь поделиться с большим числом читателей своим отзывом, публикуют его на нескольких сайтах. Также существует спам, когда в целях рекламы или антирекламы в один шаблон вписывают различные объекты. Все это нужно определять и не допускать попадания в результаты поиска.

Выбор предстоял из двух классов алгоритмов: синтаксических (шинглирование) и лексических (I-Match). Синтаксические алгоритмы локальны, т.е. для вычисления сигнатуры документа, по которой определяется схожесть с другими документами, достаточно лишь самого документа. Лексические алгоритмы глобальны, т.е. для вычисления сигнатуры требуется некая статистика по коллекции. Как следствие, для работы синтаксических алгоритмов достаточно одного прохода по данным, для лексических - либо два прохода (сбор статистики и само определение нечетких дублей), либо поддержка этой статистики в хранилище или еще где-то, что влечет усложнение манипуляций с данными (пересчет при любых изменениях).

Несмотря на то, что I-Match работает в принципе быстрее (операции, совершаемые над документами, проще и их меньше), нежели шинглирование, на реальных данных оба алгоритма работают по времени одинаково - это происходит из-за того, что все упирается в ограничение ввода-вывода при получении данных (в реальных задачах эти алгоритмы I/O-Bounded).

Поскольку шинглирование устойчиво обрабатывает небольшие изменения документов (а именно это преобразование чаще всего встречается в отзывах), нежели I-Match, а для его работы требуется один проход по данным, то выбор пал на синтаксический алгоритм. При реализации использовалась модификация с супершинглированием для уменьшения объема потребляемой памяти.

3.2 Архитектура

На рис. 3.1 представлена архитектура разрабатываемой платформы. Платформа имеет декомпозицию по функциональности. Физически каждому компоненту соответствует процесс, в целях надежности и/или производительности работающий на нескольких машинах. Это требование вносит дополнительные ограничения и сложности в дизайн компонентов и их связей. Далее будет подробно рассмотрен каждый компонент системы, его функции, детали и особенности реализации.

Storage

Один из основных компонентов платформы - хранилище. Ядром хранилища является KDepot - подсистема хранения полуструктурированных данных, работающая поверх реляционной СУБД. В этой же базе данных хранится служебная информация, необходимая для корректного обновления данных (на ее основе происходит слияние вновь пришедших данных с существующими). Дополнительно хранятся некоторые структурированные данные - информация об источниках, различные конфигурационные данные, результаты работы некоторых алгоритмов. Манипулирование данными происходит при помощи компонента Storage Driver. Он предназначен для сокрытия деталей реализации хранилища, схемы БД и предоставляет простой интерфейс для поставщиков и потребителей данных. Неформально этот интерфейс состоит из следущих методов:

- Сохрани данные

  От сохраняемых данных требуется наличие одного выделенного атрибута - URL страницы, с которых эти данные собраны. Этот URL выбран в качестве внешнего первич-

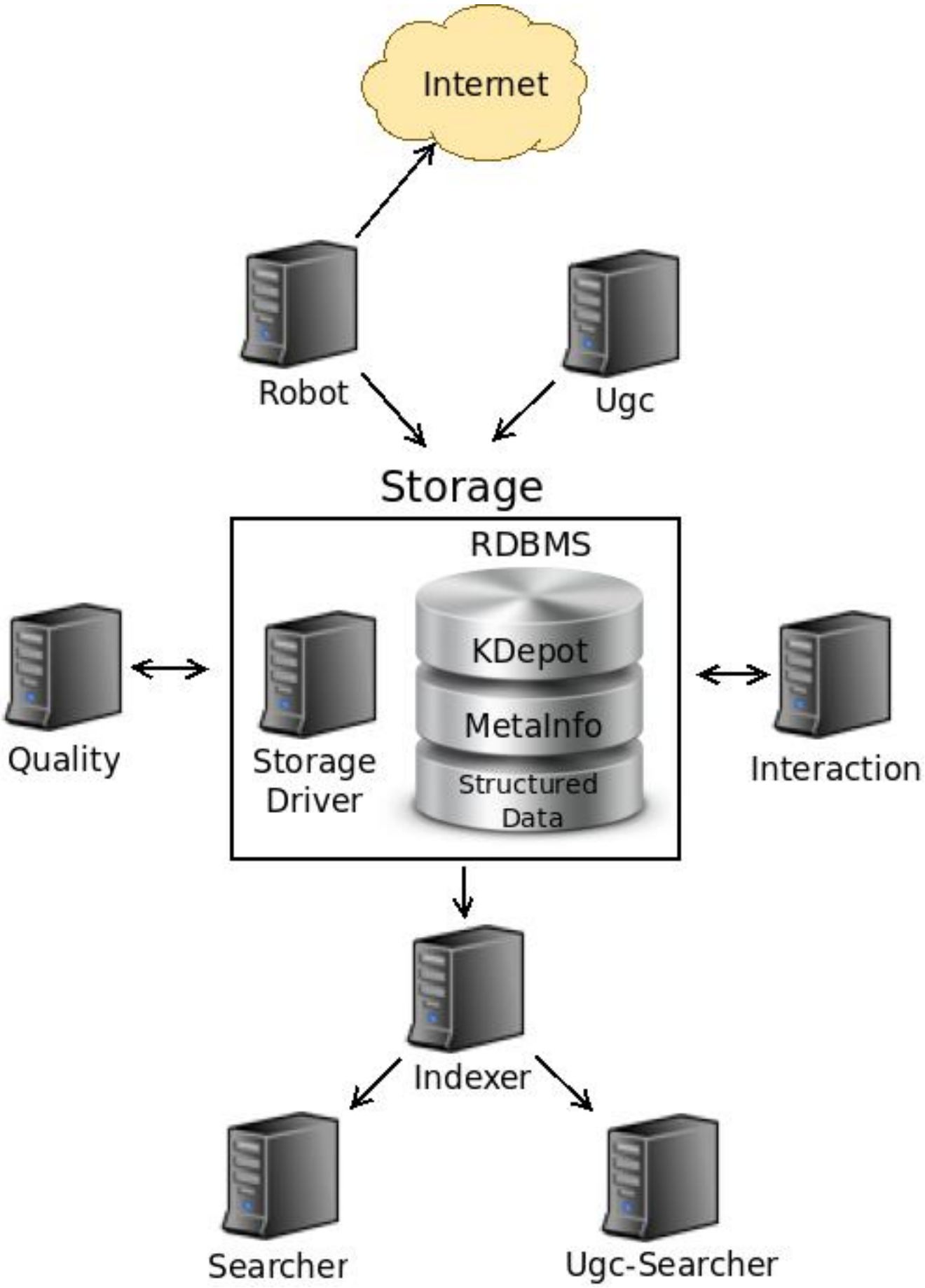

Рис. 3.1: Архитектура платформы

ного ключа. Дальше драйвер все сделает сам: при отсутствии этих данных в системе они будут созданы, при изменении - обновлены, если изменений не было - обновлена временная метка последнего прихода данных в систему.

- Верни данные по заданному ID
- Верни все данные определенного типа
- Верни данные определенного типа, изменившиеся с заданного момента времени

Как ни странно, таких четырех методов оказывается достаточно для работы платформы.

Robot

Компонент Robot является основным поставщиком данных в хранилище. В этом компоненте сосредоточена система извлечения данных из веба.

Ugc

Компонент предоставляет API поверх протокола HTTP для публикации отзывов на севрвисах. Совместно с компонентом Ugc-Searcher представляет CRUD-приложение. Процесс Ugc выполняет функции добавления, редактирования, удаления, а Ugc-Searcher - чтения. Поскольку Ugc - компонент платформы, которая используется из различных сервисов, каждый из которых имеет свой формат публикуемых данных, требуется возможность гибкого конфигурирования этих форматов. Ugc предоставляет такую возможность - правила отображение передаваемых сервисом POST- или GET-параметров в полуструктурированные данные задаются при помощи конфигурационных файлов. По сути, в этих файлах описываются правила "укладки" значений параметров HTTP-запроса (плоские данные) в древовидный объект требуемого формата, который дальше будет сохранен в KDepot. Благодаря такой возможности конфигурирования формата публикуемых данных достигается простота подключения новых сервисов к использованию платформы.

Quality

Quality - служебный компонент, предназначенный для задач, связанных с качеством данных. В частности, в этом процессе отрабатывает алгоритм, находящий нечеткие дубликаты, а также различные автоматические проверки по качеству собираемых данных.

Interaction

Компонент предназначен для выполнения задач, связанных со взаимодействием с соседними сервисами Яндекса. Одна из основных задач - связывание отзывов с объектом, на который он оставлен. В общем случае, задача понимания того, на что оставлен отзыв, достаточно сложная. Но благодаря наличию "по-соседству" тематических сервисов, таких как Яндекс.Авто, Справочник организаций и многих других, имеющих большие базы знаний по специализированным предметным областям, отзыв удается связать с объектом сервиса (в общем случае, проставить некоторый внутренний ID сервиса). При этом точность связывания практически стопроцентная, а полнота колеблется в районе 92%.

Indexer

Индексатор один из самых сложных компонентов платформы. Предназначен для создание поискового индекса по собранным данным. Индекс бывает двух видов - полный и дельта. Полный создается сравнительно редко (раз в сутки), зато дельта - очень часто (минуты). Индексатор работает с двумя видами данных (для двух разных потребителей). В индексы для Searcher попадают все данные, прошедшие все проверки по качеству и имеющие все необходимые атрибуты. В индексы для Ugc-Searcher попадают все данные, публикуемые на собственных сервисах. В свою очередь, в рамках каждого вида создаются отдельные индексы для разных типов отзывов, поскольку эти данные совершенно независимы (соответственно, и создавать их можно независимо). Таким образом, как только в системе появляются новые данные и проходят подготовку, через несколько минут они оказываются на поисковых машинах и доступны широкой аудитории.

Как говорилось выше, в качестве поискового решения по объективным причинам был выбран Apache Lucene. Каждое поколение индекса состоит из индекса самого Lucene, предназначенного непосредственно для поиска, а также из файлового key-value хранилища, содержащего полуструктурировнные данные, являющиеся результатом поиска. Существует множество key-value хранилищ, но в данном случае была выбрана реализация KDepot поверх файловой системы, что хорошо согласуется с хранением данных данных при помощи этой технологии (удобно и красиво с точки зрения реализации, когда в различных частях системы используются одни и те же представления данных). О том, какие преимущества дает хранение данных, по которым не производится поиск, вне индекса Lucene, будет рассказано в разделе, посвященном компоненту Searcher.

Исходя из того, что индексатору приходится работать с различными типами отзывов

(полуструктурирированные данные, отличающиеся друг от друга внутренней структурой), предназначенными для использования различными сервисами, как следствие, поиск и фильтрация этих данных по различными параметрам, а также требований простоты и дешевизны подключения нового сервиса (попадания на поисковые машины новых типов отзывов), была разработана подсистема, позволяющая легко и гибко настраивать индексирование различных типов отзывов. Основная идея заключается в введении правил преобразования исходных полуструктурированных данных в поисковый документ (сохраняемый в индексе Lucene) и результат поиска (сохраняется в сопутствующем key-value хранилище). Каждое правило описывает, откуда взять значение из исходных данных (путь в дереве), по какому пути положить в поисковый результат, и различные модификаторы: стоит ли осуществлять поиск по этому полю (в таком случае, значение кладется в поисковый документ), допустим ли диапазонный запрос, необходимость значения (если оно отсутствует, данные в индекс не попадают) и некоторые другие. В качестве значения, которое кладется кладется в индекс, может быть не просто значение в исходных данных, а значение указанной функции, вычисленной от исходных данных и другие похожие расширения. Для каждого типа отзыва задается набор таких правил, этот набор удобно записывать в виде XML и хранить в специальном репозитории. При очередной сессии индексации, индексатор получает эти конфигурации из репозитория и применят их при обработке данных. Применение такого подхода позволяет изменить результаты отдаваемых поисковыми машинами данных или добавит новый тип данных путем простого редактирования конфигурационных файлов. За счет этого достигается простота и скорость подключения новых сервисов.

Searcher

Searcher предназначен для поиска данных и возвращения результата в структурированном виде (XML или JSON), которые будут отображены тем или иным способом. Как говорилось выше, серчер работает с поколениями индекса, каждое из которых состоит из поискового индекса Lucene и key-value хранилища, содержащего данные, которые будут отданы в качестве результата поиска. На первый взгляд кажется, что так как документ Lucene поддерживает произвольное число дополнительных полей, нет необходимости в стороннем хранилище (все данные можно сохранить в документ Lucene). Такой подход может быть применен в случае небольших поисковых нагрузок (единицы запросов в секунду). При больших нагрузках может происходить падение из-за нехватки памяти. Дело в том, что поисковый модуль Lucene поддерживает кеширование для ускорения поиска, устроенное таким образом, что на

каждый поток отводится свой кеш. При большом количестве потоков обращений к индексу, объем памяти, занимаемой кешами, может не уложиться в отведенный процессу объем. Для этого требуется ограничивать число потоков, одновременно обращающихся к индексу, при этом блокируя другие потоки, обращающиеся к этому индексу. При этом, чем дольше работает поток, получивший доступ к индексу, тем дольше заблокированы ждущие своей очереди потоки. Основываясь на результатах профилирования, при получении документа из индекса с большим количеством дополнительных полей, значительная масса времени уходит на их начитывание. Получается, что потоки заблокированы и ждут свой очереди на чтение индекса в тот момент, когда другие потоки уже выполнили поиск и вычитывают дополнительные поля из документов. Чтобы ускорить время обращения потоков к индексу, как следствие, уменьшить время простоя других потоков в очереди на чтение к этому индексу, все данные, по которым не осуществляется поиск и которые требуются для формирования результата, выносятся в отдельное key-value хранилище. В итоге, от обращения к индексу Lucene требуется получить только набор ключей (как правило, это числа), которые быстро начитываются и с этими ключами обратиться к key-value хранилищу для получения нужных данных. Таким образом, критический ресурс в роли которого выступает поисковый индекс захватывается на меньшее время (в разы), что существенно ускоряет поиск в целом.

Ugc-Searcher

Процесс, осуществляющие поиск исключительно по отзывам, опубликованных на собственных сервисах. Внутри используются те же механизмы, что и в Searcher. Но есть небольшие изменения. Поскольку в CRUD-связке с процессом Ugc процесс Ugc-Searcher выполняет функцию Read, проще говоря, при оставлении отзыва требуется тут же показать пользователю оставленный отзыв, это влечет некоторые усложнения. Поскольку Ugc-Searcher работает с индексом, который обновляется с периодичностью в минуты, а показать сохраненный отзыв нужно мгновенно, вводится дополнительное звено. При публикации, редактирования или удалении отзыва пользователем, после соответствующих модификаций в KDepot в некотором хранилище высокой доступности сохраняются пометки о проделанных пользователем операциях. По сути, эти пометки хранят информацию о модификациях данных, которые еще не доступны в индексе. За сохранение этих пометок отвечает процесс Ugc. Ugc-Searcher при выполнении Read (показа пользователю его отзывов), сначала проверяет наличие пометок о модификациях, за измененными данными происходит обращение в хранилище, за остальными - в индекс, результаты сливаются и возвращаются в ответе процесса. Поскольку

публикация выполняется сравнительно редко (десятки раз в день), то получение данных из хранилища вполне обоснованно (нагрузка совершенно отсутствует). При создании следующего поколения индекса для Ugc-Searcher и его раскладки на поисковые машины, индексатор снимает пометки о модификациях. В качестве хранилища модификаций можно использовать как memcached (распределенное key-value хранилище данных в оперативной памяти), так и Redis (распределенное key-value хранилище, но с возможностью сохранения данных на диск), или любое другое, которых существует множество.

3.3 Реализация

3.3.1 Общие особенности реализации

Основным языком реализации платформы является Java с использованием Spring (`http://www.springsource.org`) в качестве управляющего фреймворка. Spring оказался предпочтительнее различных реализаций J2EE по следующим соображениям:

- Малая связь бизнес-модели с API фреймворка

  Использование Enterprise Java Beans предполагает создание бизнес-объектов, привязанных к фреймворку, что сильно связывает бизнес-модель и различные API используемого фреймворка. Spring спроектирован так, чтобы приложения, созданные с его помощью, зависели от наименьшего возможного числа его API. Большинство бизнес-объектов в Spring-приложениях никак не зависят от Spring. Он может сделать выбор EJB вопросом реализации, а не решающим фактором архитектуры приложения. С использованием Spring возможно реализовать бизнес-интерфейсы как POJO (Plain Old Java Object, то есть простые объекты, а не Java Beans и т.п.) или локальные EJB, никак не влияя этим на вызывающий код.

- Способствование хорошему программированию

  Уменьшение стоимости программирования до интерфейсов, вместо классов, практически до нуля.

- Простота проведения юнит-тестирования

  Поскольку бизнес-объекты являются POJO (чистый Java-код), не требуется дополнительных средств для покрытия юнит-тестами бизнес-логики приложения.

Одним из важных аспектов работы приложения является способ доступа к данным. Несмотря на то, что основной объем данных - это полуструктурированные данные, рабо-

та с которыми происходит при помощи специализированных средств, часть данных все же хранится в структурированном виде (реляционных таблицах). Это различные метаданные, настройки и прочая служебная информация. Несмотря на то, что J2EE предлагает спецификацию JPA (Java Persistence API) - технологии, обеспечивающей объектно-реляционное отображение Java-объектов и предоставляющей API для сохранения, получения и управления такими объектами, а также наличие различных реализаций JPA (основные - Hibernate, Oracle TopLink, Apache OpenJPA). Но поскольку работа со структурированными данными носит вспомогательный характер, то использование технологии JPA необоснованно усложняет разработку. Исходя из этих соображений, в качестве механизма доступа к данным был выбран Spring JDBC - часть фреймворка, отвечающая за взаимодействие с реляционным базам данных. Spring JDBC API выносит утомительную и подверженную ошибкам обработку исключений из кода приложения во фреймворк. Он берет на себя всю обработку исключений, а код приложения может сконцентрироваться на работе с БД и обработке результатов запросов, а также предоставляет осмысленную иерархию исключений, использование которой делает код более качественным.

3.3.2 Контроль качества

Разработка большого программного проекта не может обходиться без контроля качества и тестирования. На протяжении всей разработки широко применялось модульное тестирование (Unit Testing). Его использование способствовало написанию хорошо структурированного кода, уменьшению связей и зависимостей в системе, что положительно сказывается на внутренней архитектуре проекта. Также осуществление модульного тестирования упрощает проведение рефакторинга, который необходим в крупных системах. В качестве фреймворка для модульного тестирования применялся JUnit.

Несмотря на многочисленные преимущества модульного тестирования, оно не позволяет избежать всех ошибок, в частности - интеграционных проблем. Для этого применялась непрерывная интеграция (Continuous Integration). Были настроены регулярные автоматические сборки проекта. Благодаря применению такого подхода проблемы интеграции выявляются и исправляются быстрее, также обеспечивается регулярное выполнение модульных тестов. В качестве системы непрерывной интеграции была использована TeamCity от JetBrains.

Для контроля качества использовались не только автоматические средства и подходы. Присутствие экспертов в командах также положительно повлияло на качество продукта. При выборе того или иного решения учитывалось мнение экспертов, регулярно проводились

code-review, что положительно сказалось на качестве кода.

Глава 4

Внедрение

На данный момент разработанная платформа внедрена в сервис Яндекс.Авто, также внедряется на некоторых других сервисах. В случае с Я.Авто поиском по отзывам пользуется порядка 30 тысяч человек в день. При этом количество запросов различных типов колеблется от 10 запросов в секунду (10 rps) - поиск по отзывам, до 100 rps - показ средних рейтингов и других агрегированных значений на различных страницах сервиса. Запросы обрабатываются процессами на небольшом кластере машин. По результатам нагрузочных тестирований имеется десятикратный запас «прочности» , что позволит подключаться новым сервисам без изменений в архитектуре.

Глава 5

Заключение

В ходе дипломной работы были рассмотрены различные подходы и средства сбора данных из сети Интернет, хранения полуструктурированных данных, их обработки и организации поиска, проведен анализ их особенностей, выявлены достоинства и недостатки.

Спроектирована, реализована и введена в эксплуатацию масштабируемая платформа, осуществляющая сбор отзывов различных типов, их хранение, обработку и подготовку для поиска. Платформа предоставляет API, позволяющий публиковать отзывы и осуществлять поиск по собранным данным. Возможность гибкой настройки процесса индексации обеспечивает простоту и скорость ввода новых типов отзывов в систему, как следствие, подключение новых клиентов к платформе. На текущий момент решение используется в сервисе Яндекс.Авто, попутно идет внедрение в другие сервисы. Платформа имеет высокие показатели надежности и скорости работы.

Литература

1. User generated content, social media, and advertising - an overview, April 2008.
2. Denilson Barbosa, Philip Bohannon, Juliana Freire, Carl-Christian Kanne, Ioana Mano-Lescu, Vasilis Vassalos, and Masatoshi Yoshikawa. Xml storage. 2009.
3. Robert Baumgartner, Wolfgang Gatterbauer, and Georg Gottlob. Web data extraction system.
4. A. Broder, S. Glassman, and M. Manasse. Syntactic clustering of the web. proc. of the 6th international world wide web conference. 1997.
5. A. Chowdhury. Duplicate data detection. 2002.
6. D. Fetterly, M. Manasse, and M. Najork. A large-scale study of the evolution of web pages. 2003.
7. Wolfgang Meier. exist: An open source native xml database.
8. Dallan Quass, Jennifer Widom, Roy Goldman, Kevin Haas, Qingshan Luo, Jason McHugh, Svetlozar Nestorov, Anand Rajaraman, Hugo Rivero, Serge Abiteboul, Je Ullman, and Janet Wiener. Lore: A lightweight object repository for semistructured data.
9. Hui Yang and Jamie Callan. Near-duplicate detection by instance-level constrained clustering.
10. Зеленков Ю.Г. Сегалович И.В. Сравнительный анализ методов определения нечетких дубликатов для web-документов. 2007.