Санкт-Петербургский государственный университет
математико-механический факультет
кафедра системного программирования

# Создание системы хранения и выдачи данных

Курсовая работа студента 445 группы
Константинов Александр Сергеевич

Научный руководитель
Старший преподаватель кафедры СП Лопатин А.С

Санкт-Петербург
2010 год

# Оглавление

## Постановка задачи

Задачей данной курсовой работы является реализация системы хранения данных выдерживающей высокие нагрузки в режиме реального времени. Идеей для создания такого приложения послужила система mail.ru-ответы.

Mail.ru-ответы – это система, в которой хранится несколько миллионов вопросов, которые одни пользователи задают другим. Также там содержится сейчас уже более 200 миллионов ответов на эти вопросы.

По сути, идея этого сервиса очень проста: пользователь заходит на сайт, задает свой вопрос. Далее он может заходить и смотреть, не ответил ли кто на его вопрос. Если есть ответы, то он может выставлять оценку ответу по некой шкале. Также любой пользователь может не задавать вопрос, а просто запустить поиск по ключевым словам. Возможно, такой вопрос уже задавали до него.

Таким образом, в системе должен быть реализован ещё и полнотекстовый поиск – поиск по ключевым словам и, возможно, частям слов.

Как платформа выбран не отдельный сайт, а приложения Vkontakte.ru. Это упрощает разработку, т.к. не нужно хранить информацию о пользователях и не нужно проводить аутентификацию – запрос имени пользователя и пароля. Таким образом, платформа приложений Vkontakte.ru упрощает разработку интерфейсной части приложения и позволяет уделить больше внимания серверу, где и будет реализована система.

Также заранее должна быть предусмотрена архитектурная возможность разделения хранения данных на несколько серверов. Очевидно, что вначале сервер будет только один, но если количество вопросов превысит допустимый порог, то задержка будет слишком велика. Поэтому в дальнейшем разделение на несколько серверных машин должно быть простым.

Таким образом, мы получаем задачу: реализовать приложение для платформы Vkontate.ru, в котором пользователь сможет задавать вопросы, давать на них ответы, иметь возможность поиска по вопросам и ответам, а также, возможно, иметь какие-то другие дополнительные возможности, такие как поиск вопросов друзей, оценка ответов. При этом пользователь не должен испытывать неудобств при использовании этого сервиса, которые могли бы возникнуть из-за больших задержек.

Возможно, в дальнейшем платформа Vkontakte.ru уже не будет использоваться сервисом. Приложение будет запущено как отдельный продукт (сайт), поэтому стоит уделить внимание возможности комбинирования различных платформ и возможности создания своей.

Есть ещё одно требование к системе: интерфейс приложения должен быть отделен от вычислительной части, чтобы можно было изменять внешний вид приложения, но при этом не требовалось перезапускать каждый раз всё приложение. Выходит, что должна быть какая-то промежуточная выдача данных, которые бы уже и обрабатывались для выдачи пользователю.

## Введение

Социальные сети в текущий момент являются самыми посещаемыми ресурсами сети Интернет. Поэтому создание различных приложений для них является крайне выгодным. В данный момент Vkontakte.ru является самым посещаемым сайтом в России (http://www.liveinternet.ru/rating/ru/meeting/day.html). Примерно 18 миллионов человек каждый день посещает этот ресурс. И очевидно, что в дальнейшем эта цифра только вырастет.

Это означает, что посещаемость приложений в социальных сетях также велика. И если создавать аналоги каких-то сервисов вне социальных сетей, то они быстро станут конкурентоспособными за счёт того, что людям нравится, когда можно пользоваться всеми сервисами на одном ресурсе, не покидая его.

Надо заметить, что разработка сервисов для социальных сетей накладывает некоторые ограничения. Например, правила конкретной социальной сети или предоставляемое ей API. В большей степени это всё-таки относится к монетизации продукта. В данной курсовой работе эта тема не будет затронута.

Также стоит обратить внимание на то, что практически любой продукт, создаваемый для какой-то социальной, будет обладать высоким уровнем посещаемости. Поэтому разработчики должны заранее продумывать этот аспект и максимально ускорять обработку данных. При этом меньше времени уделять другим аспектам продвижения приложения, как реклама и раскрутка. Таким образом, программист может полностью сконцентрироваться на создании приложения, нежели на чём-то другом, не относящемся к этому.

Стоит отметить, что данный сервис является востребованным в сети Интернет, поскольку обычные поисковые системы не могут дать ответы на все интересующие пользователя вопросы. Поисковые системы осуществляют поиск только по ресурсам, размещенным в интернете, но если человек, положим, хочет получить совет или помощь какую-то, например, в установке какого-то софта, то такую информацию он вряд ли сможет получить в поисковой системе.

Пользователь мог бы использовать какие-то тематические сервисы: например, сайт, где профессионалы помогают устанавливать софт, но по каждой тематике создать свой сервис – крайне проблематично. Поэтому лучшее решение в данной ситуации – получение ответа от других пользователей.

В данный момент подобный сервис реализован у некоторых компаний, но в каждой реализации или платформе, на которой он размещен, есть свои минусы.

# 1. Обзор существующих технологий и подходов

С точки зрения современного пользователя, сервисы поиска ответов на интересующие их вопросы должны быть интерактивными, обладать большой базой вопросов и ответов, быть быстрыми и удобными для использования.

Для начала рассмотрим, какие подобные сервисы сейчас существуют. Какие в них недостатки по сравнению с реализуемым.

## 1.1. Существующие сервисы

В текущий момент многие крупные компании осознали, что поисковые системы не могут дать ответ на все вопросы, которые интересуют пользователя.

Самые крупные подобные сервисы сейчас предоставляют компании Google.com и Mail.ru. У этих сервисов есть свои минусы.

Первый минус состоит в том, что посещаемость этих сайтов гораздо меньше посещаемости социальных сетей. Как говорилось выше, люди не хотят покидать пределы одного ресурса. Поэтому реализация подобного сервиса для социальной сети будет востребовано.

Второй минус заключается в неудобстве того, что пользователь должен специально регистрироваться или проходить аутентификацию на их сайте. Социальная же сеть сразу же предоставляет сервису данные о пользователе, который зашёл в приложение. Таким образом упрощается взаимодействие с пользователем: он не должен вводить пароли и заполнять анкеты, но при это приложение имеет о нём всю информацию, какую нужно.

Также очевиден плюс, который есть у приложения, размещенного внутри социальной сети: возможность социализировать сервис. Т.е. добавить просмотр вопросов друзей, отображать вопросы, которые интересуют друзей. Таким образом, востребованность сервиса будет ещё выше.

Ещё один пример подобного сервиса: «вопросы» в сети Vkontakte.ru. Здесь уже отсутствуют те минусы, которые были перечислены выше, но присутствуют свои.

Основной минус состоит в том, что вопросы скорее всего увидят только те пользователи, которые состоят в друзьях с создателем вопроса. Поэтому вероятность получить ответ минимальна. Также отсутствуют какие-либо дополнительные возможности: оценка вопросов и ответов, лучшие вопросы, последние вопросы. Вопросы никак не разделяются по тематикам. Поэтому использование данного сервиса крайне неудобно и нерационально, т.к. вряд ли можно получить ответ на интересующий вопрос.

Кроме того этот сервис достаточно давно не предлагает пользователям ничего нового и не имеет никакой обратной связи. Что плохо, т.к. пользователи часто предлагают какие-то новые идеи, которые улучшат качество продукта.

Таким образом, существуют сервисы, которые будут являться серьёзными конкурентами разрабатываемому. Но выигрыш данного в том, что он находится внутри социальной сети и не уступает по возможностям другим, уже реализованным. То есть единственное, что нужно добиться – сделать его столь же высокопроизводительным, как у конкурентов. Также необходимо заранее продумать некоторые социальные функции, чтобы конкурировать с самой социальной сетью.

### 1.2. Технологии

Существует несколько технологий, используя которые можно было бы добиться того или иного результата. Но по многим объективным причинам они не могут обеспечить сервис нужными свойствами. Рассмотрим несколько их них.

#### 1.2.1. Хранение в памяти

Проще всего, конечно, данные просто хранить в памяти сервера. Для этого можно использовать средства, предоставляемые языком программирования.

Очевидно, что такой подход удобен тем, что разработчик сам решает, в каком формате что хранить, а также как осуществлять поиск по данным. Также ощутимый плюс в том, что скорость работы такого хранилища будет высокой.

Самым существенным минусом такого подхода является то, что невозможно хранить все вопросы в оперативной памяти, т.к. её размеры ограничены. Другой минус – это то, что программист сам вынужден реализовывать поиск по данным, а также все интерфейсы.

Поэтому такая технология никоим образом не решает поставленной задачи, кроме как в тестовой версии сервиса.

#### 1.2.2. База данных

Очевидно, что база данных на начальных этапах развития приложения полностью удовлетворяет всем потребностям. В ней можно делать выборку по тем или иным параметрам, делать поиск по вопросам, а также достаточно удобно хранить все данные.

Большой плюс такого подхода – это простота реализации и взаимодействия с базой. В данный момент существует достаточно много разновидностей баз данных. Поэтому можно выбрать ту, которая наиболее удобна.

К сожалению, при больших нагрузках база данных уже не сможет удовлетворять всем потребностям. Если в базе будет хотя бы более 10 тысяч вопросов, то при полнотекстовом поиске задержка будет слишком ощутимой. Если же вопросов будет ещё больше, то база данных просто не сможет выдержать такую нагрузку.

### 1.2.3. Библиотека для полнотекстового поиска

Вторым подходом к данной задаче могла бы являться библиотека, которая индексирует необходимые данные, а далее очень быстро выполняет поиск по построенному индексу.

Как и в подходе с базой данных, тут присутствует простота в разработке и взаимодействия. Стоит признать, что использовать библиотеку и получать из неё данные несколько сложнее, чем из базы данных, т.к. данные чаще всего хранятся в виде строк и отсутствуют всевозможные функции, упрощающие работу с данными. Всё же это уже готовое решение и использовать его удобно.

Также скорость поиска какого-то вопроса по ключевым словам будет крайне быстрой, даже если в базе будет сотня тысяч вопросов.

Рассмотрим плюсы конкретно библиотеки Lucene (http://ru.wikipedia.org/wiki/Lucene):

- скорость индексации свыше 20MB в минуту на Pentium M 1.5GHz
- требуется малый объем RAM — «heap» всего 1MB
- ранжированный поиск — лучшие результаты показываются первыми
- множество мощных типов запросов: запрос фразы, wildcard запросы, поиск интервалов и т. д.
- поиск основанный на «полях» (таких как, заголовок, автор, текст)
- возможность сортировать по различным полям
- multiple-index поиск с возможностью объединения результатов

Заметим, что при использовании библиотеки Lucene большим плюсом является то, что используется малое количество оперативной памяти и тогда её можно использовать для чего-то другого.

Таким образом, эта библиотека предоставляет быстрый и удобный поиск данных при минимальных ресурсных затратах.

Большим минусом такого подхода является то, что требуется время на построение индекса. Поэтому если мы хотим добавить какие-то данные, то потребуется время на то, чтобы перестроить индекс. А это может занимать даже более нескольких минут. Поэтому

при каждом добавлении вопроса делать перестроение индекса не представляется возможным.

Отсюда получается вывод, что данный подход был бы идеален, если бы данные не добавлялись в базу.

### 1.3. Выводы

Был приведен ряд подобных сервисов и перечислены их недостатки. У каждого из сервисов внутри социальной сети есть большое преимущество в посещаемости и востребованности. Также с течением времени становится очевидным, что всё большая часть населения регистрируется в той или иной социальной сети, а следовательно довести до них приложение будет достаточно просто и не затратно.

Сильную конкуренцию могли бы составить сами социальные сети, но они не занимаются глубоким развитием подобных сервисов. Другие же разработчики приложений пока что не сделали ничего подобного, поэтому эта область пока что остается свободной для внедрения. Поэтому реализация подобного сервиса одним из первых будет крайне выгодным и развивающимся проектом.

Существуют простые способы реализации подобных систем, которые дают успех на начальных уровнях развития проекта. Если рассмотреть технологии перечисленные выше, то становится очевидно, что их использование по отдельности не принесёт желаемого результата, поскольку у каждой из них есть минусы, которые критичны при большом количестве вопросов.

Можно было бы использовать, например, базу данных для начальных этапов развития проекта, но скорость роста посещаемости и объемов данных будет велика, поэтому задержки станут ощутимыми уже на первых неделях существования приложения. Таким образом, стоит реализовывать какую-то систему, которая сразу же сможет выдерживать много запросов.

Хоть все приведенные выше технологии и не способны удовлетворить всем потребностям, но при этом каждая из них предоставляет некие возможности, комбинируя которые можно получить высокопроизводительную систему.

## 2.Реализация

### 2.1. Архитектура клиент-серверного приложения

Разработанный в рамках курсовой работы сервис включает в себя программное обеспечение сервера и клиентскую часть. Сервер функционирует в среде Ubuntu 9.0.1, но может быть также успешно запущен и под любой другой, для которой реализована виртуальная машина Java.

Клиентская часть сервиса не будет рассмотрена, т.к. это набор простых скриптов на языке JavaScript и они могут быть заменены на что-либо ещё, т.к. по сути, просто отображают пользователю те страницы, которые передал сервер, а также передают некую дополнительную информацию серверу, как список друзей пользователя или информацию о нём.

Из рис 1. видно, что взаимодействие с сервером могут осуществлять не только пользователи из сети Vkontakte.ru, но и другие пользователи. По сути, база вопросов одна и для того чтобы добавить другую социальную сеть или просто реализовать отдельный сервис на своём уже сайте, нужно только немного добавить функционала на сервер в виде нового типа юзеров и каких-то небольших других в виду особенностей того или иного ресурса. К примеру, можно добавить поддержку пользователей из сети «Мой мир», созданной компанией Mail.ru. Таким образом, становится понятным, что клиентская часть может быть практически любой, важно именно то, как обрабатывает данные сервер и в каком виде выдает результаты.

Рис 1.

В рамках курсовой работы была разработана выдача результата в формате HTML, т.к. такой формат отображается абсолютно всеми браузерами и не требует установки какого-либо дополнительного программного обеспечения. В свою очередь, можно написать клиентскую часть на языке ActionScript и создать приложение на Flash.

При входе пользователя в приложение клиентская часть отсылает серверу всю информацию о пользователе и запрашивает титульную страницу. Далее просто реагирует на действия пользователя, пересылает его запрос на сервер и подгружает результаты, которые передаёт сервер.

Сейчас это реализовано в виде небольшого JavaScript кода, который заменяет содержимое одного из div'ов.

### 2.2. Архитектура сервера

Серверная часть приложения реализована на языке Java. В качестве основы для сервера, т.е. для обработки запросов выбрана свободнораспространяемая библиотека Jetty - контейнер сервлетов, написанный полностью на Java, который может использоваться как HTTP-сервер (http://ru.wikipedia.org/wiki/Jetty).

Этот сервер умеет распараллеливать запросы и выполнять их параллельно в несколько потоков. При поступлении запроса сервер обрабатывает его и передает управлению пользовательскому Handler'у, передав ему все необходимую информацию о запросе.

Создание сервера производится с помощью кода, приведенного ниже.

```
final Server server = new Server();
final Connector connector = new SelectChannelConnector();
connector.setPort(port);
connector.setHeaderBufferSize(HEAD_BUFFER_SIZE);

final QueuedThreadPool threadPool = new QueuedThreadPool();
threadPool.setMaxThreads(maxThreadPoolSize);

final HandlerCollection handlerCollection = new HandlerCollection();
handlerCollection.setHandlers((Handler[]) handlers.toArray());

server.setConnectors(new Connector[]{connector});
server.setThreadPool(threadPool);
server.setHandler(handlerCollection);
server.start();
```

Т.е. сначала создается коннектор, которому указывается порт, по которому будет производиться подключение, а также размер буфера. Далее создается пул потоков, в котором будут храниться потоки, выполняющие обработку запросов. После этого создается коллекция обработчиков запросов. И в самом конце создается сам сервер и запускается.

Чаще всего в примерах по созданию HTTP-сервера на Jetty, которые можно найти в интернете, указывается несколько другой код для создания сервера, но опытным путем было выяснено, что данный код лучше, т.к. скорость обработки запросов будет выше, т.к. настройка количества потоков, а также использование пула потоков, выполнено более качественно.

После того, как сервер на Jetty выполнил начальную обработку запроса, управление передается пользовательскому коду. На рис 2. показан дальнейший процесс обработки запроса.

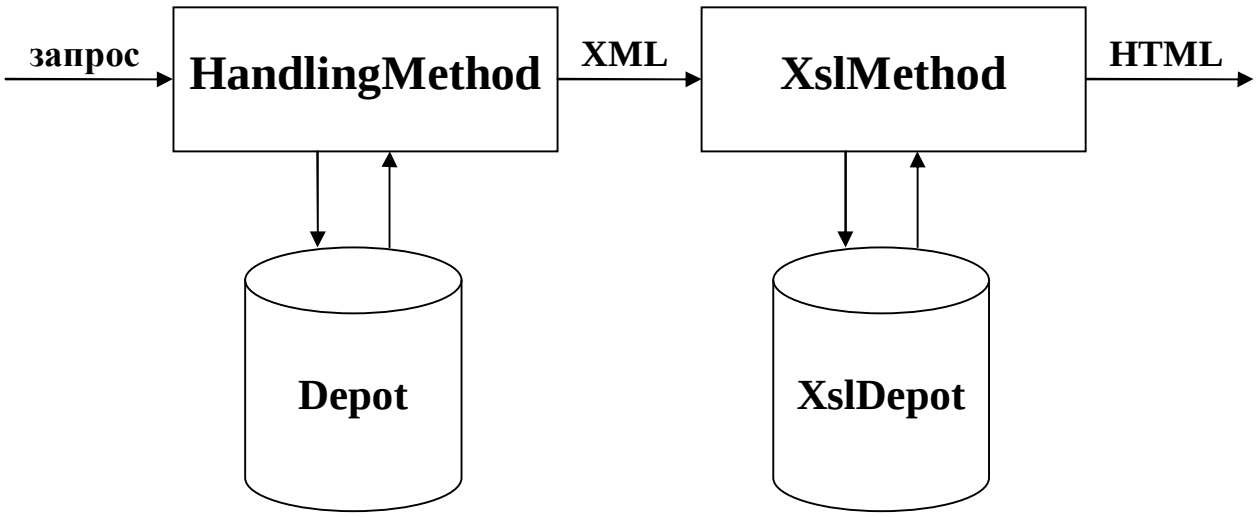


Рис 2.

Сначала клиентская часть приложение посылает запрос на сервер, чтобы отобразить ту или иную страницу. Далее сервер узнает метод, который хочет вызвать клиент, находит обработчик этого метода и передает ему управление.

Некий HandlingMethod получает управление и в зависимости от своего назначения делает те или иные действия. Например, выбирает 10 последних вопросов из базы или добавляет туда новый. При этом идёт обращение к хранилищу данных, добавляются или выбираются вопросы и ответы.

После того, как HadlingMethod выбрал все данные и сформировал ответ, он преобразует все данные с помощью XSTream в формат XML. XSTream – это библиотека для сериализации Java-объектов в XML и обратно (http://xstream.codehaus.org/). Эта библиотека очень удобна, т.к. сама при помощи reflection-api узнает поля классов. Единственное, что требуется настоить в этой библиотеке: передать сокращения полных путей классов, т.е. сделать alias, чтобы в XML не было такого: <ru.spbgu.qanda.data.Question>, а было просто <question>. После создания XML управление передается в XslMethod.

Для преобразования из xml в html используется язык xslt (***eXtensible Stylesheet Language Transformations***) (http://ru.wikipedia.org/wiki/XSLT). Этот язык позволяет быстро получить html и при этом получается отделение представления от данных. В любой момент можно быстро изменить xsl, но не трогать код на Java, поэтому перезагрузка сервера не потребуется.

Класс XslMethod обращается к хранилищу XslDepot и по названию метода получает xsl-код. Далее он накладывает этот xsl на xml и получается готовый html, который уже и передается клиентской части.

### 2.3. Depot

Как было указано ранее, использовать по отдельности базу данных, библиотеку для индексации или хранилище в памяти не представляется возможным. Поэтому было решено скомбинировать их.

На рис 3. показано, каким образом и в каком порядке данные передаются между ними.

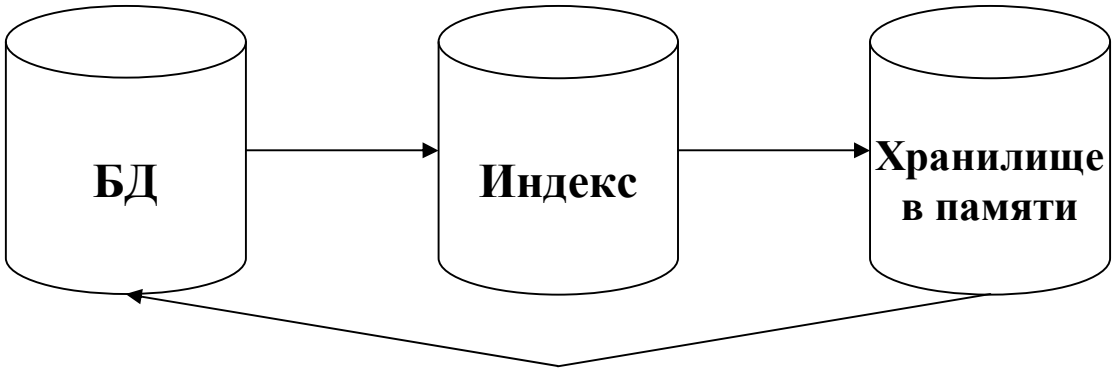


Рис 3.

При запуске сервера вопросы и ответы из БД полностью выгружаются, по ним создается индекс. Таким образом, если нужно что-то найти, то выполняется поиск в индексе Lucene, что происходит достаточно быстро даже при больших объёмах данных.

Когда пользователь добавляет свой вопрос или же отвечает на какой-то вопрос, то эти данные нужно внести в базу и в индекс. Но выше было указано, что изменять индекс слишком затратно по времени. Поэтому данные, которые пользователь добавил, сначала хранятся в памяти сервера. При этом при накоплении какого-то количества данных, происходит batch-insert в базу (т.е. вставка в базу в виде одной транзакции).

При запросе на какие-либо данные, поиск по ним происходит сначала в памяти. Если же в памяти данные не найдены или они недостаточны, то идёт обращение к индексу.

Очевидно, что при таком подходе данные в памяти будут очень быстро накапливаться и оперативная память закончится. Поэтому раз в несколько минут добавление в базу

останавливается, начинается выгрузка всех данных в новый индекс. В тот момент, когда новый индекс на жестком диске создан, происходит подмена пути у хранилища в индексе, поэтому данные там становятся актуальными. После чего происходит очистка хранилища в памяти, т.к. эти данные уже есть в индексе, оставляются только данные, которые добавили уже после начала пересоздания индекса.

В тот момент, пока запрещено добавление в базу, сервер всё равно отвечает на запросы, т.к. база данных не нужна при выдаче результатов.

Таким образом, сервер постоянно находится в рабочем состоянии и нет задержек при ответе на запрос.

То есть, база данных – это долговременное хранилище данных, в котором вопросы и ответы хранятся постоянно и где не осуществляется поиск. Индекс – здесь как раз и происходит поиск данных, но сюда данные не добавляются в режиме реального времени. Хранилище в памяти же хранит разницу между базой данных и существующим индексом. Т.е. данные в базе – это данные в индексе плюс данные в памяти.

Благодаря такой системе хранения все минусы каждого типа хранения данных становятся несущественными, остаются только плюсы. У БД – это удобство использования и надежность долговременного хранения, у индекса – высокая скорость доступа и поиска по данным, у хранилища в реальном времени – высокая скорость доступа, поиска и возможность изменения прямо в памяти.

### 2.3.1. База данных

В качестве базы данных была выбрана база Mysql. Причиной выбора её послужило то, что она бесплатна и быстро работает, при этом не требует много аппаратных ресурсов.

Единственное что отличает работу с базой в приложении, где много данных, в том, что нельзя выбирать из базы все вопросы или ответы одним запросом. Когда происходит индексация данных, то из базы нужно выгрузить абсолютно все данные. Поэтому сначала из базы выбирается список всех id, т.к. он помещается в оперативной памяти. Далее список id разбивается на части по какому-то количеству (сейчас там выставлено число 500). После чего делаются выборки уже конкретных списков вопросов по известным id. После чего к каждому списку вопросов, выгруженному из базы, применяется метод react, который объявлен в интерфейсе EntitiesReactor. В свою очередь EntitiesReactor уже что-то делает с этими вопросами, например, записывает их в индекс, как было указано выше.

Таким образом, система, которая выгружает данные из базы, не будет тратить слишком много ресурсов, т.к. не будет пытаться разместить сразу все вопросы в памяти, а будет это делать последовательно.

Вставки в базу делаются также сразу же порциями. Для этого используется стандартный метод batchUpdate.

### 2.3.2. Индекс

Как было сказано выше, для индексации выбрана библиотека Lucene. Для индексации создан класс Indexer.

Т.к. индексировать нужно вопросы, ответы и, возможно, что-то ещё в дальнейшем, то было желание написать какой-то общий класс, а не свой класс для каждого типа данных. Поэтому все данные должны реализовывать интерфейс Depotable, а также для каждого из них должна быть Factory. Внутри интерфейса Depotable описаны методы, с помощью которых, можно представить класс с данными как набор из пар ключ-значение, где и ключ, и значение – строка. Factory же наоборот создаёт класс из такого набора. Также благодаря интерфейсу Depotable индексер получает дополнительную информацию: какие поля нужно индексировать, а какие просто сохранить, т.к. по некоторым полям поиск осуществляться не будет.

В класс Indexer передается коллекция из объектов и он превращает их во внутренний формат, понятный библиотеке Lucene. Также этот класс умеет выполнять поиск по каким-то критериям и выдавать коллекцию объектов на запрос.

### 2.3.3. Хранилище в памяти

Реализация хранилища в памяти отличается от всего, что было описано до этого, т.к. до этого надо было просто умело воспользоваться готовым решением, здесь же нужно реализовывать всё с нуля.

Для хранения данных в памяти используется HashMap, в котором по значению id можно получить вопрос. Но т.к. также нужно было реализовать ещё и поиск по параметрам, то поиск по тексту и поиск по численным параметрам разбиты на две независимых части. Внутри класса Question хранятся поля, по которым будет осуществляться поиск.

Поиск по числовым данным может быть 5 типов: меньше, меньше или равно, равно, больше или равно, больше. Чтобы не осуществлять поиск перебором была выбрана структура данных, в которой поиск можно осуществить за логарифм: это деревья.

Т.к. в памяти всё хранится в виде ссылок на объекты, то не слишком затратно по объёму оперативной памяти дублировать ссылки. Поэтому по каждому параметру, по которому нужно делать поиск, создается дополнительное дерево. Ключи которого – значение параметра, а значения – список из записей, у которых такое значение параметра.

Далее, когда делается запрос на поиск по нескольким параметрам, алгоритм использует некие эвристики для ускорения и работает следующим образом: по каждому параметру делается поиск, если в какой-то момент результат оказывается пустым, тогда поиск сразу же прекращается. Далее результаты сортируются по количеству записей, и начинается пошаговое пересечение начиная с самых коротких. Т.к. если окажется, что в какой-то момент результат пуст, то дальше можно не пересекать уже. Очевидно, что чем два списка короче, тем выше вероятность того, что результат окажется пустым.

Для хранения используются стандартные средства языка Java: TreeMap. В них поиск по и отсечение части дерева осуществляется за время пропорциональное логарифму от количества записей.

Далее же, когда два списка объединяются, они сортируются по id, чтобы осуществить пересечение за один проход.

Для того, чтобы списки не нужно было сортировать по id, можно было бы хранить всё в немного другой структуре: декартово дерево - это двоичное дерево, которое являются двоичным деревом поиска по первому ключу и двоичной кучей по второму ключу (http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE). Это означает, что если будет производиться поиск по параметру, то для получения результата в отсортированном по id порядке потребуется время пропорциональное размеру результата.

Но этот подход не был выбран в виду того, что в памяти данных будет храниться не очень много. Поэтому при таких объёмах данных разница между линейным временем и линейным временем, умноженным на логарифм, будет несущественна. В свою очередь, реализация декартовых деревьев не очень удобна, а также при внесении нового функционала они могут оказаться несовместимыми. Поэтому был выбран более простой подход в силу объективных причин.

### 2.4. Обновление

Описанная выше структура приложения хорошо работает, если требуется добавлять и выбирать данные. Но случается так, что данные меняются, например, текст вопроса или какие-то ещё данные.

Если пользователь добавил вопрос достаточно давно и вопрос уже попал в индекс, то основная проблема в том, что данные в индексе нельзя поменять. Поэтому пришлось придумать алгоритм, который помог бы избежать ошибок при выдаче данных.

Вначале было реализовано так, что если данные изменяются, то они просто выгружаются из индекса и хранятся в памяти. Т.е. если далее осуществляется поиск по данным, то они находятся сначала в хранилище в памяти и выдаются пользователю.

Но стало очевидным, что может получиться так, что при каком-нибудь запросе нужно будет объединить данные из индекса и данные из памяти. Тогда произойдёт ошибка, т.к. будут выданы старые данные из индекса.

Тогда было решено пытаться объединять результаты с проверкой на повторы и убирать более старую версию данных. К сожалению, такой подход также оказался ошибочным.

Положим, пользователь изменил текст вопроса с «Мой вопрос» на «Чужой вопрос». Тогда если нужно будет осуществить поиск по словам «Мой вопрос», то тогда при выдаче результатов не будет 2 вопроса выдано, т.к. в памяти хранится «Чужой вопрос» и этот текст не удовлетворяет критериям поиска. А значит, не будет происходить проверки на то, что данные устарели, и будет выведен ошибочный результат.

Чтобы бороться с этим, пришлось ввести список измененных сущностей, в котором хранятся все id записей, которые были изменены. Далее при запросе к индексу каждый раз производится проверка на то, были ли данные какие-нибудь изменены.

По сути, получилось так, что хранилище в памяти – это и есть разница между индексом и базой данных, как и планировалось сделать изначально.

### 2.5. Разделение на несколько машин

Заранее была продумана возможность дальнейшего разделения хранилища на несколько серверов, т.е. кластеризация. Кластеризацию возможно будет произвести по id пользователей, поскольку при запросе чаще всего используется именно id пользователя, т.к. чаще всего отображаются либо просто последние вопросы, либо добавляются новые, а тогда id пользователя либо вообще не важен и последние вопросы можно просто хранить отдельно, либо же известен id пользователя и тогда заведомо известно, на какой сервер нужно переправить запрос.

Но всё это будет производиться только в том случае, если один сервер не будет справляться с количеством запросов.

### 2.6. Дополнительные возможности

Изначально приложение планировалось разработать для социальных сетей, поэтому предусматривался некоторый список функций, который будет у приложения.

Например, одна из таких функций – это отображение списка вопросов друзей. Можно было бы хранить список друзей каждого юзера, а потом делать выборку по его друзьям, но было решено подойти к этому вопросу с другой стороны.

Было посчитано, что в среднем у человека в социальной сети примерно 150-200 друзей. Если тратить на хранение каждого друга по 4 байта, то в сумме получится всего около килобайта. Т.е. можно просто поместить эту информацию в каждый вопрос: при индексации добавить в каждую запись поле «список друзей» пользователя, которому принадлежит сущность.

Далее если пользователь хочет увидеть список всех вопросов друзей, то в индексе просто происходит поиск по его id и сразу же выдается ответ, не нужно делать запросы на поиск вопросов каждого друга. Таким образом, сильно экономится время на поиск при небольших затратах на лишнее место на жёстком диске (на индекс). Такая оригинальная оптимизация помогла сильно ускорить работу дополнительной функции.

Другая же дополнительная функция – оценка вопроса. Она реализована точно так же, как и текст в вопросе, т.е. пользователь выставляет оценку, при этом считается, что произошло изменение самого вопроса, т.е. применяется алгоритм по изменению записей, описанный в предыдущем параграфе.

## Заключение

В рамках данной курсовой работы был разработан сервис, выдерживающий высокие нагрузки. Изначально было очевидно, что самым «тонким» местом приложения будет хранилище вопросов и ответов. Поэтому ни одна из существующих в свободном распространении библиотек не подходила для этого. Было реализовано собственное хранилище данных, которое хорошо масштабируемо, быстро выполняет поиск и выдачу результата, позволяет вводить дополнительные функции, надежно хранит существующие данные. Хранилище реализует функции добавления, выборки, изменения и полнотекстового поиска. В большинстве случаев от хранилища не нужны никакие больше функции, кроме описанных.

Была поставлена цель реализовать сервис, который не будет уступать своим конкурентам по скорости работы и возможным функциям, и это задача была выполнена.

В дальнейших планах – провести небольшое тестирование реализованной системы, попробовать вычислить «узкие» места, а также минусы текущего хранилища. На данный момент все минусы, которые были у технологий, описанных в начале, у текущего хранилища отсутствуют, т.е. в него может поместить большое количество данных, при этом поиск будет быстрым, не будет никаких задержек в моменты пересоздания индекса.

Далее планируется запустить этот проект в социальной сети Vkontakte.ru. Для этого требуется доделать внешнее оформление – обратиться за помощью к профессиональному дизайнеру, далее нужно написать на языке xslt код по переводу и xml в требуемый html по созданному дизайну. Также требуется внести некоторые дополнительные функции, связанные конкретно с этой социальной сетью. Но все эти задачи уже не разработчика, а скорее менеджера, т.к. нужно найти подходящего дизайнера и верстальщика и сделать им заказ. Таким образом, можно считать, что программная часть сервиса выполнена.

Далее планируется проследить начальное развитие сервиса, внести некоторые оптимизации и улучшения в коде. Также необходимо будет внести некоторые дополнительные функции, чтобы улучшить «юзабилити» приложения.

Возможно, что часть дальнейшего развития помогут придумать сами пользователи, т.к. у приложения будет обратная связь и пожелания пользователей будут учитываться в дальнейшем развитии проекта.

Если проект будет расти, то потребуется кластеризация и покупка серверов, но все эти проблемы будут решаться уже по ходу, т.е. тогда, когда появятся, т.к. в архитектуру изначально заложена возможность разделения хранилищ, поэтому это всё будет не так сложно реализовать.

Также есть желание сделать этот сервис отдельным от какой-либо социальной сети: это хочется сделать для того, чтобы при запросах к обычным поисковым системам иногда происходил переход к текущему сервису. К сожалению, страницы внутри социальных сетей не индексируются поисковыми системами, поэтому требуется реализовать свой сайт.

В архитектуре серверной части уже заложена возможность подключения пользователей не из социальной сети Vkontakte.ru: введены типы, т.е. у каждого пользователя есть тип, поэтому вынесение сервиса на отдельную платформу потребует только реализации страницы регистрации пользователя и изменения анкетных данных, а также небольших изменений в оформлении.

Таким образом, сервис реализован практически полностью, после небольшого тестирования и оформления будет запущен и будет произведено тестирование уже в реальных условиях. План развития в дальнейшем также уже сформирован, сроки зависят только от востребованности системы пользователями и от скорости роста проекта.

## Список источников

[1] Н.Вирт. Алгоритмы и структуры данных. : Мир, 1989 //Декартово дерево, 326с

[2] А. Валиков. Технология XSLT. : БХВ-Петербург, 2002

[3] Extesible Markup Language // http://www.w3.org/XML

[4] JavaScript Tutorial // http://www.w3schools.com/js/default.asp

[5] Jetty Codehaus Wiki // http://docs.codehaus.org/display/JETTY/Jetty+Wiki

[6] MySQL 5.1 Reference Manual // http://www.mysql.ru/docs/mysql-man-5.1-en/

[7] XSL Transformations // http://www.w3.org/TR/xslt

[8] XStream Tutorial // http://xstream.codehaus.org/tutorial.html