Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Марийский государственный университет»

Институт цифровых технологий

Физико-математический факультет

Кафедра прикладной математики и информатики

ДЕМО ВЕРСИЯ ДИПЛОМНОЙ РАБОТЫ

на тему: «Разработка нереляционной распределённой базы данных типа ключ-значение, c поддержкой поисковых запросов на основе модели MapReduce.»

Научный руководитель:

кандидат пед. наук

Кондратенко И. Б.

Выполнил: студент ФМФ ПИ–43

Ф.В. Авдеев

Йошкар-Ола

2021

Содержание

[Введение 3](#_Toc73931662)

[1 Теоретическая часть 5](#_Toc73931663)

[1.1 История развития СУБД 5](#_Toc73931664)

[1.2 Архитектура СУБД 6](#_Toc73931665)

[1.3Методы хранения данных 9](#_Toc73931666)

[1.4 Способы доступа к данным 10](#_Toc73931667)

[1.5 MapReduce 13](#_Toc73931668)

[2Практическая часть 17](#_Toc73931669)

[2.1 сетевой интерфейс I/O 17](#_Toc73931670)

[2.2 Организация данных 18](#_Toc73931671)

[2.3 Транспортный уровень 18](#_Toc73931672)

[2.4 Прикладной уровень 21](#_Toc73931673)

[2.5 Алгоритм векторных часов 23](#_Toc73931674)

[2.6 Тестирование 26](#_Toc73931675)

[Заключение 27](#_Toc73931676)

[Список литературы 28](#_Toc73931677)

[Приложение 29](#_Toc73931678)

# Введение

Сегодня трудно представить какую-нибудь информационную систему, которая не имела в качестве основы (или важной составляющей) базу данных. Концепции и технологии баз данных складывались постепенно и всегда были связанны с развитием систем автоматизированной обработки информации. Однако это уже вполне сложившаяся дисциплина, более инженерная, чем научная, основана на достаточно формальных подходах и включающая широкий спектр приемов и методов создания баз данных. Системы управления базами данных должны обеспечивать сохранение данных в течение длительного времени, а также возможность выборки и актуализации. Данные создаются, для того, чтобы они могли, были быть использованы в каком либо процессе. Хотя пути извлечения и характер использования могут быть разные.

Базы данных, достаточно медленно развивающие направление. Элементная база ЭВМ, парадигмы программирования меняются быстрее, чем хранимые данные теряют актуальность, поэтому создатели баз данных должны уметь интегрировать в создаваемую систему данные, находящиеся под управлением устаревшей СУБД и уметь создавать системы, чтобы вновь созданные данные могли быть, наследованы следующим поколением систем и разработчиков.

Широкое использование баз данных различными категориями пользователей привело к созданию удобных интерфейсов требующие минимум времени на освоение управление системой и на создание мощных, гибких СУБД с широкими возможностями и с развитой защитой данных от разрушения случайного или преднамеренного. Появились и средства автоматизации разработки БД, позволяющие создавать БД пользователю даже не владеющего Основам теории БД.

Для создание практической полезной БД разработчик должен разбираться в области концепций и технологий СУБД, должен понимать прикладные задачи пользователя БД, так же должен подумать о качестве описание и представление этих решений чтобы пользователь мог использовать, сопровождать и развивать систему после разработчика.

Объектом изучения в этой работе являются нереляционые распределенные БД, повышение эффективности с помощью поисковых запросов на основе модели MapReduce.

Данная тема актуальна, так как большинство web – приложения имеют

в составе своей архитектуры БД которые нужно расширять, поддерживать. Такие БД должны быть быстро отвечать на запросы пользователей, они должны одновременно обрабатывать несколько задач.

Предмет исследования нереляционые распределенные БД и поисковые запросы на основе модели MapReduce.

Цель исследования разработать нереляционыу распределенные БД с поисковыми запросами на основе модели MapReduce и протестировать.

Задачи

Структура курсовой работы

# Теоретическая часть

## 1.1 История развития СУБД

Наука о баз данных достаточно старая и консервативная она потерпела наименьшее количество изменений по сравнению с большинства информационных наук.

Примерно до середины шестидесятых годов файл использовался в качестве БД. В таких БД информация часто разрушалась из-за невозможности или с затруднением одновременной работы и недостаточного поиска. С середины шестидесятых годов до 1980 начали использовать нереалиционные базы данных. Было понятно, что использовать только файл было накладно для производства и были разработаны иерархические системы, причём у потомка может быть только один предок, далее появились сетевая модель, было логическая продолжение иерархической модели, тут у потомка могло быть более одного предка. У данных моделей были свои достоинства и свои недостатки.

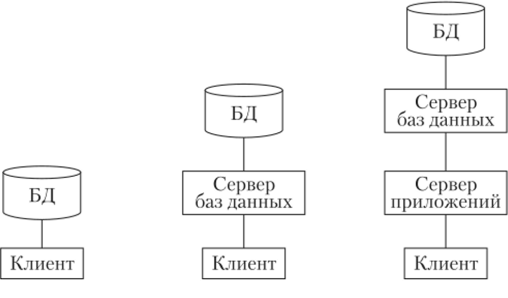
В 1970 году британский учёный Эдгар Кодд выпустил работу “A Relational Model of Data for Large Shared Data Banks”. Данная работа считается первым трудом по  реляционному хранению данных. После выпуски данной работы, начали активно разрабатывать систему по реляционному хранению данных.

Начало восьмидесятых годов – выпуск реляционных СУБД.

В данное время активно разрабатываются NewSQL базы данных, которые наследуют реляционную структуру и семантику, но построены с использованием более современных, масштабируемых конструкций, развиваются Объектно-Ориентированные базы данных и NoSQL это нереляционные БД, огромный общий класс БД в котором все модели отличаются друг от друга разными характеристиками.

## 1.2 Архитектура СУБД

По своей архитектуре СУБД делятся на одно-, двух- и трехзвенные модели (Рисунок 1). В однозвенной архитектуре используется единственное звено (клиент), обеспечивающее необходимую логику управления данными и их визуализацию. В двухзвенной архитектуре значительную часть логики управления данными реализует сервер баз данных (сервер БД), в то время как клиентское звено в основном занято отображением данных в удобном для пользователя виде. В трехзвенных СУБД используется промежуточное звено — сервер приложений



А

Рисунок 1 – архитектуры СУБД

В зависимости от местоположения отдельных частей СУБД различают локальные и сетевые СУБД. Все части локальной СУБД размещаются на компьютере пользователя, обращающегося к базе данных. Чтобы с одной и той же БД одновременно могло работать несколько пользователей, каждый пользовательский компьютер должен иметь доступ к своей копии локальной БД. Существенной проблемой СУБД такого типа является синхронизация содержимого копий данных (репликация данных), именно поэтому для решения задач, требующих совместной работы нескольких пользователей, локальные СУБД не пригодны.

К сетевым относятся файл-серверные, клиент-серверные и распределенные СУБД. Непременным атрибутом этих систем является сеть, обеспечивающая аппаратную связь компьютеров и делающая возможной совместную работу множества пользователей с одной и той же базой данных.

В файл-серверных СУБД вся база данных обычно размещается на одном или нескольких запоминающих устройствах достаточно мощной машины, специально выделенной для этих целей и постоянно подключенной к сети. Такой компьютер называется файл-сервером. Безусловным достоинством СУБД этого типа является относительная простота ее создания и обслуживания, так как фактически все сводится лишь к организации локальной сети и установке на подключенных к ней компьютерах сетевых операционных систем. Между локальными и файл-серверными вариантами СУБД нет особых различий, так как в них все части СУБД сосредоточены на компьютере пользователя. По архитектуре они обычно являются однозвенными, но в некоторых случаях могут использовать сервер приложений. Недостатком файл-серверных систем является значительная нагрузка на сеть. Например, если пользователю, работающему на клиентском компьютере, нужно отыскать сведения об одной из книг, имеющихся в библиотеке, то по сети вначале передается весь файл, содержащий сведения обо всех книгах, и лишь затем в созданной таким образом локальной копии данных отыскиваются нужные сведения. При интенсивной работе с данными нескольких десятков пользователей пропускная способность сети может оказаться недостаточной, и пользователя будут раздражать значительные задержки в реакции СУБД на его требования. Файл-серверные СУБД могут успешно использоваться в относительно небольших организациях с количеством клиентских мест до нескольких десятков.

Клиент-серверные (двухзвенные) системы значительно снижают нагрузку на сеть, так как клиент общается с данными через специализированного посредника — сервер БД, который размещается на машине с базой данными. Сервер БД принимает запрос от клиента, отыскивает в данных нужную запись и передает ее клиенту. Таким образом, но сети передаются относительно короткий запрос и единственная нужная запись, даже если база данных содержит сотни тысяч записей. Как правило, запрос к серверу формируется на специальном языке запросов SQL, поэтому часто серверы БД называются SQL-серверами. Серверы БД представляют собой относительно сложные программы, разрабатываемые различными фирмами, например: Microsoft SQL Server (SQL Server) производства корпорации Microsoft, Sybase Adaptive Server корпорации Sybase, Oracle производства одноименной корпорации, DB2 корпорации IBM, InterBase корпорации Borland и т.д. Клиент-серверные СУБД обеспечивают функционирование, или масштабируются, до сотен и тысяч клиентских мест.

Распределенные СУБД могут содержать несколько десятков и сотен серверов БД. Количество клиентских мест в них может достигать десятков и сотен тысяч. Обычно такие СУБД обеспечивают работу организаций государственного масштаба (например, Центральной избирательной комиссии РФ), отдельные подразделения которых рассредоточены на значительной территории. В распределенных СУБД некоторые серверы могут дублировать друг друга с целью достижения предельно малой вероятности отказов и сбоев, которые могут исказить жизненно важную информацию.

Актуальность распределенных СУБД возросла в связи со стремительным развитием Интернета. Опираясь на возможности Интернета, распределенные системы строят не только организации государственного масштаба, но и относительно небольшие коммерческие предприятия, обеспечивая своим сотрудникам работу с корпоративными данными на дому и в командировках.

В нашей программе мы будем использовать клиент-серверная модель так-как в этой модели большая часть работы выполняется серверной частью при минимуме нагрузки на клиента. Это позволит разграничить доступа к разным уровням информационной системы, каждому клиенту свой уровень доступа. Уменьшение нагрузки на сеть ввиду того, что клиент в основном передаёт серверу команды, а тот уже их исполняет. Так же каждый клиент сможет работать с ресурсами сервера вне зависимости от используемой операционной системы. Недостатки же данной модели если выйдет из строя сервер то приведет к неработоспособности всей системы, его использующей, высокая нагрузка на серверное оборудование и канал связи до него. Но это можно предотвратить при распределении серверной работы на несколько компьютеров, другими словами собрать распределенную систему вычислений.

## Методы хранения данных

Различные БД хранят структурируют и хранят данные по разному, мы сейчас рассмотрим основные метод: реляционные, графовые, ключ-значение.

Реляционные БД хранят данные в таблицах со столбцами и строками, данные хранятся в полях таблицы, осуществляется взаимодействие с таблицами ща счёт реляционной алгебры. Пример БД использующие данные метод: MySql, PostgreSql, Visual FoxPro, Access.

Графовые представляют данные в виде вершин графа, а взаимосвязь данных в виде рёбер соединяющие эти вершины. Вершины могут быть направленными и не направленными. Обход графа в графовой базе данных можно выполнять либо по определенным типам ребер, либо по всему графу. Пример БД использующие данные метод: FlockDB, Giraph, HyperGraphDB, IndraDB, InfiniteGraph.

Модель “ключ-значение” предназначенная для хранения, извлечения и управления ассоциативными массивами, структура данных, более известная сегодня как словарь или хеш-таблица. Словари содержат коллекцию объектов или записей, которые, в свою очередь, содержат множество различных полей, каждое из которых содержит данные. Осуществляется взаимодействие с БД по ключу.

В нашей БД будем использовать метод хранения данных “ключ-значение” плюсы данного метода он легко расширяем, можно изменять структуру бд и добавлять данные при развитии проекта, так же обрабатывают данные как одну непрозрачную коллекцию, которая может иметь разные поля для каждой записи. Это обеспечивает значительную гибкость и более точно следует современным концепциям, таким как объектно-ориентированное программирование. Данные в нашей БД будут хранится в формате “json”.

## 1.4 Способы доступа к данным

В простых моделях “клиент – сервер”, в роле встроенного ПО для доступа к данным выступает ODBC драйвера. Обычно в более сложных системах размещается промежуточное звено SQL – шлюз, который даёт набор API. Обычно создаётся такое приложение на языках высокого уровня.

Каждая СУБД имеет библиотеку доступа и набор драйверов СУБД для разных ОС. Библиотека доступа включает набор функций, которые позволяют клиентской части соединиться с БД, передавать запросы на сервер и получать данные. Такой способ очень гибкий, но есть недостатки:

• Разработка тока под конкретную ОС и на том языке которая поддерживает библиотека.

• Драйвера БД определяет допустимые типы сетевых интерфейсов.

• Функции в библиотеках обычно не унифицированы.

ODBC – Open Database Connectivity. Нужен для унификации доступа к данным, хранящиеся на удаленных серверах. ODBC – использует спецификацию CLI. За доступ к отдельной СУБД отвечает драйвер. Пользовательская часть этого не видит. Поэтому приложение в большей степени становится не зависимой от БД.

OLE DB – Object Linking and Embedding Date Base – Встраивание и связывание объектив в базах данных. Обеспечивает единый интерфейс доступа к данным через провайдера. В отличие от ODBC, OLE DB даёт общее решение обеспечение COM – приложеним доступ независимо от источника информации. Он содержит два главных компонента, это провайдер данных и потребитель данных. Потребитель (клиент) – это приложение или COM – компонент, общающий через API к OLE DB. Провайдер (сервер) – это приложение отвечающий на вызовы OLE DB и возвращающий обьект.

ADO – Active Data Object – это высокоуровневый универсальный интерфейс к OLE DB. Эта модель не содержит таблиц, различные среды и БД. Здесь основными компонентами являются, объект соединение, который создаёт связь с провайдером, объект набор данных и объект Команда – выполнение процедуры или SQL – строки. Можно рассматривать ADO как язык программирования операций с БД и так–как он опирается на OLE DB, то можно использовать в любых приложениях Microsoft.

Один из способов доступа к данным это использование API, но это означает зависимость создаваемой программы от используемой СУБД и при переходе на другую систему это влечёт за собой переписывание кода клиентской части. Следующий этап обеспечения доступа к данным клиентского приложения является создание универсального механизма доступа к БД, дающий клиентскому приложению стандартный набор функций, классов, служб, которые нужны для работы с разными базами данных. Эти компоненты должны располагаться в библиотеках. Такие библиотеки называются драйверами или провайдерами баз данных.

Наиболее популярными механизмами доступа к данным (Universal Data Access, UDA) являются:

• ODBC.

• OLE DB.

• ADO.

• BDE.

Первые три по сути являются промышленными стандартами. Последний долгий период времени был единственным решением для доступа к данным, используемый в средствах разработки компании Borland.

Так же есть набор определенных правил, по которым желательно строить веб приложение. REST (Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. В это стиль входит шесть правил:

* Клиент-сервер. Отделяя пользовательский интерфейс от хранилища данных, мы улучшаем переносимость пользовательского интерфейса на другие платформы и улучшаем масштабируемость серверных компонент за счёт их упрощения.
* Отсутствие состояния. Каждый запрос от клиента к серверу должен содержать в себе всю необходимую информацию и не может полагаться на какое-либо состояние, хранящееся на стороне сервера. Таким образом, информация о текущей сессии должна целиком храниться у клиента.
* Кэширование. Это ограничение требует, чтобы для данных в ответе на запрос явно было указано - можно их кэшировать или нет. Если ответ поддерживает кэширование, то клиент имеет право повторно использовать данные в последующих эквивалентных запросов без обращения на сервер.
* Единообразие интерфейса. Если применить к системам инженерный принцип общности/единообразия, то архитектура всего приложения станет проще, а взаимодействие станет прозрачнее и понятнее. Для выполнения этого принципа необходимо придерживаться нескольких архитектурных ограничений. REST накладывает на интерфейс четыре ограничения: 1) идентичность ресурсов; 2) манипуляция над ресурсами через представление; 3) исчерпывающие, понятные человеку сообщения; 4) гипермедиа как движок для состояния приложения (HATEOAS) – ссылки на другие ресурсы внутри приложения.
* Многоуровневая система. Многоуровневость достигается за счёт ограничения поведения компонентов таким образом, что компоненты "не видят" другие компоненты, кроме тех которые расположены на ближайших уровнях, с которыми они взаимодействуют.
* Код по требованию (необязательно). REST позволяет наращивать функциональность клиентского приложения по мере необходимости при помощи скачивания и исполнения кода в виде апплетов или скриптов. Это упрощает клиентские приложения, уменьшая количество заранее написанных возможностей.

## 1.5 MapReduce

Проблемы распараллеливания вычислений, распределения данных и обработки сбоев заставляют отказаться от простой модели вычислений с большим объёмом сложного кода, поэтому была предложена новая абстракция компанией gooogle, которая легко позволяет производить необходимые вычисление и при этом скрывает все детали распараллеливания. Модель MapReduce применяется в множество задач, распределённый поиск и сортировку, обращение графа веб-ссылок, обработку статистики логов сети, построение инвертированных индексов, кластеризацию документов, машинное обучение и статистический машинный перевод.

На практике программы не всегда будут работать быстро, главным преимуществом такой модели является эффективное распределение данных между узлами и маленькое количество кода. Функция разделения данных и количество данных на выходе функции Map могут очень сильно влиять на производительность. Есть дополнительные модели которые помогут уменьшить количество данных, например функция Combiner. Задачи которые хорошо решаются на нераспределенных системах, а входные данные помещаются в оперативную память одного компьютера использования фреймворка MapReduce неэффективно. Так как эти фреймворки разработаны, чтобы иметь возможность восстановления целых узлов кластера во время вычислений, они записывают в распределённое хранилище промежуточные результаты работы. Такая технология окупается только тогда, когда в вычисления учувствует много компьютеров. Когда выходит из строя один компьютер проще перекинуть его задачи на другой.

Преимущество в то что операции предварительной обработки и свёртки производятся распределённо. Операции предварительной обработки работают независимо друг от друга и могут производиться параллельно. Так же и множество других узлов могут совершать свёртку, необходимо только чтобы все результаты предварительной обработки с одним значением ключа обрабатывались одним рабочим узлом в один момент времени, иногда процесс может быть менее эффективный по сравнению с последовательными алгоритмами. MapReduce должен работать на больших данных, которые последовательно не могут обрабатываться

Для задач, которые решаются быстро на нераспределённых системах, а входные данные помещаются в оперативную память одного компьютера или небольшого кластера, использование фреймворка MapReduce неэффективно

MapReduce – это модель распределенных вычислений. Она используется для параллельных вычислений над очень большими данными, вплоть до нескольких петабайт.

Работа MapReduce состоит из двух шагов: Map и Reduce, названных аналогично функциям высшего порядка из многих языков программирования, применяемым на этих шагах.

На шаге Map идёт предварительная обработка входных данных. Для этого один из компьютеров (master node — главный узел) получает входные данные задачи, разделяет их на части и передает другим компьютерам (worker node — рабочим узлам) для предварительной обработки.

Затем на шаге Reduce идёт свёртка предварительно обработанных данных. Главный узел получает ответы от рабочих узлов и на их основе формирует результат — решение задачи.

Главным преимуществом такой технологии, что она позволяет распределено производить операции предварительной обработки и свёртки. Операция предварительной обработки (Map-шаг) работают независимо друг от друга, параллельно (но на практике ограниченно количество используемых процессоров и источниками входных данных). Так же могут рабочие узлы осуществлять свертку (Reduce-шаг) — для этого нужно чтобы все результаты предварительной обработки с одним конкретным значением ключа обрабатывались одним рабочим узлом в один момент времени.

Параллелизм данного подхода даёт возможность восстановиться после частичных сбоев серверов, если в рабочем узле, производящем операцию предварительной обработки или свертки, возникает сбой, то его работа может быть передана другому рабочему узлу (при условии, что входные данные для проводимой операции доступны).

Классический пример приложения MapReduce – это программа на псевдокоде, подсчитывающая количество различных слов в наборе документов (Рисунок 2).

*// Функция, используемая рабочими узлами на Map-шаге для обработки пар ключ-значение из входного потока*

voidmap(String name, String document):

*// Входные данные:*

*// name – название документа*

*// document – содержимое документа*

**for** each word w in document:

EmitIntermediate(w, "1");

*// Функция, используемая рабочими узлами на Reduce-шаге для обработки пар ключ-значение, полученных на Map-шаге*

void reduce(String word, Iterator partialCounts):

*// Входные данные:*

*// word – слово*

*// partialCounts – список группированных промежуточных результатов. Количество записей в partialCounts и есть требуемое значение*

int result =0;

**for** each v in partialCounts:

result += parseInt(v);

Emit(AsString(result));

Рисунок 2 – Пример алгоритма на псевдокоде

В предоставленном выше коде, на шаге Map каждый документ разбивается на слова, он возращает пары “ключ – значение”, в примере ключом является ссамо слово, а значением “1”. Если одно и то же слово встречается несколько раз, то в результате будет столько же пар сколько слово встречалось в документе. Далее все пары с одинаковыми ключами обьеденяются и передеаются на вход функции *reduce*, которая складывает их для получения общего количества вхождения данного слова в документы.

Вызов функции Map распределены между неким количество компьютеров с помощью деления входных данных на набор из M частей. Входные части могут обрабатываться параллельно несколькими машинами. Вызовы функции Reduce распределены разделением промежуточного пространства ключей на R{\displaystyle R} частей с помощью функции разделения (например, hash(key)modR{\displaystyle hash(key)modR} ). Задаются пользователем количество частей R и функция разделения.

# 2Практическая часть

Разрабатывать программу будем на языке python. В разработке мы будем применять модель СУБД клиент-сервер, СУБД будет использовать два протокола данных, прикладной который будет обеспечивать взаимодействие сети и пользователя и транспортного который обеспечивать доставку данных. Всего будет восемь пакетов, шесть пакетов со стороны клиентов, это запрос на регистрацию пользователя, авторизацию, чтение данных, изменение данных, удаление данных, удаление пользователя. И два пакета от сервера, это ответ на чтение данных и ответ на все остальные запросы. Будем использовать сетевой интерфейс ввода/вывода через асинхронные селекторы, что позволит сразу нескольким клиентам подключиться к серверу. БД будет храниться в json-файле по типу ключ-значение. Доступ же к этому файлу только у авторизированных пользователей.

## 2.1 сетевой интерфейс I/O

Для реализации интерфейса мы будем использовать библиотеку “selectors”. Данная библиотека обеспечивает высокоуровневое и эффективное мультиплексирование I/O, основанное на примитивах модуля select. Пользователи поощряются при использовании этого модуля, если они не хотят точного контроля над используемыми примитивами уровня ОС.

И будем использовать библиотеку “socet”. Этот модуль обеспечивает доступ к интерфейсу сокета BSD . Он доступен во всех современных системах Unix, Windows, MacOS и, возможно, на дополнительных платформах.

Константы хранятся отдельным модулем, из которого по мере необходимости приложение использует.

С помощью библиотеки “socet” создаём объект “sock”. С помощью библиотеке “selectors” создаем объект sel стандартным конструктором “sel = selectors.DefaultSelector()” Далее подключаемся к определенному порту “ sock.bind(Conf.HOST, Conf.PORT)”. Устанавливаем нужно количество одновременных соединений “sock.lisen(100)”. Задаём неблокирующий режим сокета “sock.blocking(False)”. Отдельной функцией “Accept(sock, mask)” которая принимает два параметра сокет и маску, будем регистрировать каждого клиента и функцию “ServerLoop(con, mask)” которая обрабатывает его пакет. Внутри функции “Accept(sock, mask)” мы получаем сокет клиента и его адрес “conn, addr = sock.accept()” и регистрируем, передавая параметрами сокет клиента, битовую маска отслеживающию событие и функцию читающие принимающие данные “sel.register(conn, selectors.EVENT\_READ, ServerRecvLoop)”. В функции “ServerLoop(con, mask)” когда данные не поступают от клиента, мы отменяем регистрацию файла “sel.unregister(conn)”. Так же саму функцию “Accept(sock, mask)” регистрируем “sel.register(sock, selectors.EVENT\_READ, Accept)” где параметр “ sock ” это сокет сервера.

## 2.2 Организация данных

Данные хранятся в json файле и для организации и взаимодействии с ним написан класс “ FedorDB ” который сделан отдельным модулем. Он с помощью функции “load()” загружает файл в память как словарь и с помощью других функций может получить значения, изменять значение, удалять, может работать со всеми типами объектов в python, к примеру словари, списки, коттеджи и с примитивными типами данных. А с помощью функции “dump()” сохраняет данные в памяти в файл. На данный момент в программе хранятся данные в файле “data.json”.

## 2.3 Транспортный уровень

Пакеты на транспортном уровне ограниченны конечным числом байтов, по бокам каждого пакета выделен по одному байту с шестнадцатеричным числом “ 0x7f ” это сделано для того чтобы можно было понять где начло пакета и где конец. Далее идёт код (Таблица 1), занимает 2 и 3 байт пакета где старший бит стоит вначале для того чтобы можно было понять от клиента или от сервера пришёл пакет. После от 60-92 байта идут данные которые должны получить сервер или клиент размер зависит от передаваемого пакета, после ещё 2 байта контрольная сумма, которая формируется отправителем, а получатель должен сверить с полученную контрольную сумму и ту которую вычислил. Для контрольной суммы был написан отдельный модуль “Cheksum” в котором есть две функции, “Cheksum(data: bytes, poly=0x8408)” вычисляет контрольную сумму и возвращает, CheksumTransportPackech(packetData) сравнивает контрольную сумму с присланным пакетом и возвращает “True” – если суммы совпадают, “False” – если не совпадают. Контрольная сумма рассчитывается алгоритмом “ CRC16”.

Таблица 1 – коды пакетов

|  |  |
| --- | --- |
| Коды в пакете | Описание |
| 0x1 | запрос авторизации |
| 0x8001 | ответ авторизации, удаление данных, изменение данных, регистрацию пользователя, удаление пользователя |
| 0x2 | запрос на чтение данных |
| 0x2001 | ответ на чтение данных |
| 0x3 | запрос на удаление данных |
| 0x4 | запрос на изменение данных |
| 0x5 | запрос на регистрацию пользователей |
| 0x6 | запрос на удаление пользователя |

Таблица 2 – структура транспортного пакета

|  |  |  |
| --- | --- | --- |
| **№** | Назначение | Размерность |
|  | START – начало пакета, начинается с 0x7f | 1 байт |
| 1 | CMD – тип транспортного сообщения | 2 байта |
| 3 | DATA – данные транспортного сообщения | 60 - 92байт |
| 4 | CRC16 – (рассчитывается для полей №1,2) с полином 0x8408 | 2 байта |
|  | END – конец пакета, заканчивается с 0x7f | 1 байт |

Для Расчёта CRC16 можно использовать быстрый табличный метод:

const unsigned short Crc16Table[256] = {

0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,

0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,

0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,

0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,

0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,

0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,

0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,

0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,

0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,

0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,

0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,

0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,

0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,

0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,

0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,

0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,

0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,

0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,

0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,

0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,

0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,

0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,

0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,

0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,

0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,

0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,

0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,

0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,

0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,

0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,

0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,

0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0

};

unsigned short CRC\_16TN(U16 crc, unsigned char \*pcBlock, unsigned short len)

{

while (len--)

crc = Crc16Table[((crc>>8) ^ \*pcBlock++) & 0xFFL] ^ (crc << 8);

return crc;

}

Инициализирующее значение при расчёте 0xFFFF, пример:

CRC = CRC\_16TN(0xffff, obuf, sz);

Что значительно ускорит вычисление расчётной суммы.

## 2.4 Прикладной уровень

Данные перед отправкой представляют собой словарь, который кодируется в строки, а потом в байты. Для кодирования в нужный формат данных и декодирование был написан отдельный модуль “ProtoHelper”. Так же он для каждой ячейки данных выделяет отдельное число байт и если данные не умещаются в этот размер то он обрезает данные, а если данные меньше соответствующего размера то он заполняет нулями не занятое количество байт. Ниже приведены таблицы пакетов, каждый пакет входит в транспортный уровень в поле “data” (Таблица 2).

Таблица 3 – Пакет авторизации пользователя

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | LOGIN – логин пользователя. | 30 байт |
| 1 | PASSWORD – пароль пользователя. | 32 байта |

Таблица 4 – Пакет ответ авторизации, удаление, изменение, регистрацию, удаление пользователя.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя. | 30 байт |
| 1 | Message – ответ о провале или успехи действия. | 30 байта |

Таблица 5 – Пакет запроса на чтение данных.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя | 30 байт |
| 1 | KEYS – ключи по которым следует использовать выборку данных. | 30 байта |

Таблица 6 – Пакет ответа на чтение данных.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя | 30 байт |
| 1 | MESSAGE – словарь выборки из БД по ключам, ключ-значение. | 30 байта |

Таблица 7 – Пакет запроса на удаление данных.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя | 30 байт |
| 1 | KEYS – ключи значение которых нужно удалить. | 30 байта |

Таблица 8 – Запрос на изменение данных.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя. | 30 байт |
| 1 | KEYS – ключи значение которых нужно изменить. | 30 байта |
|  | VALUE – значение на которые нужно изменить. | 30 байта |

Таблица 9 – Запрос на регистрацию пользователей.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя | 30 байт |
|  | LOGIN – логин нового пользователя | 30 байт |
| 1 | PASSWORD – пароль нового пользователя. | 32 байт |

Таблица 10 – Запрос на удаление пользователей.

|  |  |  |
| --- | --- | --- |
| № | Назначение | Размерность |
|  | IDSESSIONS – id сессии подтвержденного пользователя | 30 байт |
|  | LOGIN – логин пользователя которого нужно удалить | 30 байт |
| 1 | PASSWORD – пароль пользователя которого нужно удалить | 32 байта |

Так же любые манипуляции с БД разрешено проводить только авторизированным пользователем. При авторизации пользователя сервер генерирует число (IDSESSIONS) и отсылает пользователю, так же сохраняет у себя на момент подключение клиента, в свою очередь клиент принимает ключ и сохранят к себе и при следующем отправке других запросов передают этот ключ серверу, где сервер сверяет переданных ключ клиента со своим сохраненным. На данный момент в программе ключи сохраняются json-файлах. “settingsBD.json” – для сервера, “settingsCL.json” – для клиента.

Пароль сервер не видит, так как на клиентской части он хэшируется алгоритмом “md5” и отправляется уже в зашифрованном виде, использована была библиотека с таким же названием. Файл логином и паролем пользователей в программе хранится в “users.json”.

## 2.5 Алгоритм векторных часов

Консистентность данных — это согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечивость. Условия целостности данных стали записывать в виде правил и ввели триггеры — процедуры, которые вызывались до и после выполнения запроса. Kонсистентность является важнейшим понятием теории управления данными (data management) и входит ACID свойства (Atomicity, Consistency, Isolation, и Durability) — Атомарность, Консистентность, Изоляция данных и Долговечность.

Согласование данных – это процесс поддержки всех реплицированных копий одного элемента данных в одинаковом состоянии при изменении одной из копий.

Репликация – это процесс поддержки нескольких копий одного элемента данных в различных локациях.

Целостность базы данных означает соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.

Непротиворечивость — Отсутствие в базах данных, либо частях одной и той же базы сведений, которые бы противоречили друг другу.

Так же в распределенных вычислительных системах есть проблема времени, невозможно гарантировать одинаковое время во всех узлах, в связи с проблемой «дрейфа часов». Для достижения адекватной степени ошибок при работе, распределенных вычислительных системах используют механизмы синхронизации времени.

Не все БД реализуют консистентность или реализуют только часть свойств консистентности, в силу более быстрой работы системы. Но в нашей БД мы реализуем. Для реализации консистентности мы выбрали алгоритм векторные часы, так как он наиболее функциональный из известных алгоритмов.

Для реализации векторных часов в распределенной системе, требуется добавлять к сообщениям информацию о локальных метках посылающего процесса. Для системы из N процессов требуется иметь массив из N логических отметок времени, одна логическая отметка на процесс. Каждый процесс, наряду со своей локальной меткой времени, поддерживает копии известных ему глобальных времен остальных процессов. Для поддержания консистентности копии применяется следующий набор правил:

• в начальный момент времени все метки нулевые;

• как только у процесса произошло внутреннее событие, он переводит свои часы вперед на единицу;

• передача сообщения — тоже событие, поэтому перед передачей любого сообщения внутренние часы тоже увеличиваются на единицу и копия всех известных процессу часов посылается вместе с сообщением;

• получение сообщения процессом заставляет его перевести все часы в своем векторе на максимум из известных ему и присланных значений.

Опишем данный алгоритм на псевдокоде:

Procedure Precedence^, р2: array[l..n] of int)

if Vp1[i] < p2[i], i ? 1 ... n then

if 31 : р2[г] < p2[i] then

return p2 предшествует p2

else

return Предшествование p2 и p2 не определено

end if

end if

if Vp2[i] < рДг], г ? 1 ... n then

if 3i : p2[i] < px[i] then

return p2 предшествует рг

else

return Предшествование p2 и p2 не определено

end if

end if

end procedure

Неформально этот алгоритм можно описать так: первое событие предшествует второму в том случае, если все значения всех отметок времени для первого события не больше соответствующих отметок времени второго события и при том хотя бы одно из значений отметок времени первого события строго меньше соответствующего значения второго события.

## 2.6 Тестирование

# Заключение

# Список литературы

# Приложение