Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра информационных и управляющих систем

Работа допущена к защите
Зав. кафедрой

_____П.Д. Дробинцев

«_____» _____
_____ 2016 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: ***Разработка системы обработки больших объёмов данных о столкновениях транспортных средств***

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент гр. 43504/3 _____ Целоусов Н.В.

Руководитель _____ Коликова Т. В.

Санкт-Петербург
2016

ЗАДАНИЕ

к работе на соискание степени бакалавра

студенту Целоусову Никите Владимировичу

1. Тема проекта (работы)

Разработка системы обработки больших данных о столкновениях транспортных средств

(утверждена распоряжением по факультету от_____ № _____)

2._____ Срок сдачи студентом оконченного проекта (работы)

3. Исходные данные к проекту (работе)
   3.1. Документация Apache Hadoop: https://hadoop.apache.org/docs/r2.7.2/
   3.2. Документация Apache Storm: http://storm.apache.org/releases/current/index.html
   3.3. Данные о столкновениях ТС
   3.5. Данные о плотности траффика
   3.6. Данные о погоде
4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)
   4.1. Обзор инструментов и методов для работы с большими данными
   4.2. Архитектура системы
   4.3. Основные функциональные алгоритмы системы
   4.4. Анализ результатов

Реферат

БОЛЬШИЕ ДАННЫЕ, КЛАСТЕР, HADOOP, PIG, STORM, R, MAPREDUCE, МАСШТАБИРУЕМОСТЬ, ПРОГНОЗИРОВАНИЕ, СТАТИСТИКА

Работа составляет 64 страницы и содержит 24 иллюстрации.

В данной работе рассматривается разработка системы обработки данных о столкновениях транспортных средств. Система служит для сбора разнообразной статистики и ее визуализации, а также для прогнозирования количества аварий в реальном времени. Система разработана с помощью технологий работы с большими данными и в соответствии с архитектурными стандартами построения таких систем, также проведен обзор этих технологий и архитектур и описаны некоторые использованные математические методы.

Результатом работы являются статистические данные, которые могут быть использованы для повышения безопасности дорожного движения и мониторинга дорожной ситуации в режиме реального времени. Система имеет высокий потенциал для дальнейшего расширения и улучшения: использование новых источников данных, повышение точности

Содержание

Список иллюстраций

Используемые определения и сокращения

SQL (structured query language) – формальный непроцедурный язык программирования, применяемый для управления данными в реляционной базе данных.

СУБД (система управления базами данных) - совокупность программных средств, обеспечивающих управление баз данных.

SAN (Storage Area Network) - архитектурное решение для подключения внешних устройств хранения данных.

HDFS (Hadoop distributed file system) – распределенная файловая система, созданная специально для Apache Hadoop.

CSV (Comma separated values) - текстовый формат, предназначенный для представления табличных данных.

GPS (Global Positioning System) - спутниковая система навигации, обеспечивающая измерение расстояния, времени и определяющая местоположение во всемирной системе координат.

UDF (User defined functions) – функции, предоставляемые пользователем, использующим ту или иную программу, которые встраиваются в эту программу, расширяя ее стандартные возможности.

DAG (Directed acyclic graph) - орграф, в котором отсутствуют направленные циклы, то есть пути, начинающиеся и кончающиеся в одной и той же вершине.

HTTP (HyperText Transfer Protocol) —протокол прикладного уровня передачи данных.

Введение

На сегодняшний день актуальной является проблема статистического анализа данных о транспортной ситуации и особенно автомобильных аварий. Ежегодно в России происходит около 200 тыс. ДТП, а за последние 10 лет в авариях погибло 350 тыс. человек и пострадало более 3 млн. В России, в отличие от других стран, таких как США, Германия или Великобритания на сегодняшний день не существует как таковой единой системы хранения и анализа данных о столкновениях. Разработка такой системы могла бы позволить выделить аварийно-опасные участки дорог, возрастные и социальные группы водителей и другие закономерности. На основе полученных данных могут быть предприняты действия, которые приведут к существенному сокращению количества ДТП и улучшения дорожной безопасности в целом. Также, в расчет могут быть взяты и другие данные, например, о погоде и плотности потока транспортных средств, с целью поиска возможных зависимостей, предсказания и быстрого реагирования на изменение дорожной обстановки.

В работе предлагается реализовать прототип системы статистической обработки таких данных, и проанализировать возможности ее применения. К сожалению, наряду с отсутствием системы обработки, в Российской Федерации не существует в открытом доступе и самих данных, поэтому система будет имплементирована на основе данных, собранных Министерством Транспорта Великобритании, находящимися в открытом доступе и содержащими данные о каждом ДТП начиная с 1979 года до сегодняшнего момента. Также для анализа будут использованы два других типа данных: о погоде и о плотности траффика.

Цель работы — создать систему, которая:

- на основе этих данных позволяет получить статистику, которая помогла бы увеличить безопасность дорожного движения
- позволяет эффективно визуализировать полученные результаты
- на основе этих данных и данных о плотности траффика сможет прогнозировать количество аварий в реальном времени
- будет являться легко расширяемой для новых источников данных

На сегодняшний день в Великобритании уже существуют системы мониторинга на основе этих данных, разработанные частными компаниями. Также ежегодно простейшие отчеты формирует само Министерство Транспорта. Однако, системы, используемые ими

- Не являются свободно распространяемыми
- Не обладают открытым исходным кодом
- Основаны на реляционных базах данных

Последний пункт может вызывать опасения для долгосрочного использования продукта, так как с увеличение объема данных, может сделать использование реляционных баз неэффективным. Как известно реляционные базы данных масштабируются хуже, чем нереляционные решения, поэтому в данной работе будут использованы технологии для работы с большими данными, а именно экосистема Apache Hadoop.

1. Обзор предметной области

1.1 Сравнение СУБД на основе SQL с Hadoop

Для решения задачи аналитики собранных архивных данных могут быть использованы разные подходы. В последнее время часто сравниваются две парадигмы: традиционные (реляционные базы данных) и распределенные системы для обработки больших данных. Несмотря на то что современные СУБД на основе SQL достаточно хорошо работают с данными объемом соответствующим поставленной задаче, данная задача будет реализована с помощью средств и инструментов, входящих в экосистему Apache Hadoop.

Сравнение СУБД на основе SQL с Hadoop:

- Структурированные данные: Одно из отличий заключается в том, что реляционные базы данных по своей архитектуре ориентированы на работу со структурированными данными, а многие приложения Hadoop имеют дело с неструктурированными данными, например, с текстовыми.
- Масштабирование: Масштабирование коммерческих реляционных баз данных обходится дорого. По своей природе они направлены на вертикальное масштабирование: чтобы развернуть более крупную базу данных необходимо приобрести более мощное оборудование. Но в какой-то момент, объем данных может возрасти настолько, что достаточно мощного сервера для его обработки не существует в принципе. Также важен тот факт, что высокопроизводительные машины слишком дороги сами по себе, а именно чаще всего машина, которая в пять раз мощнее стандартного компьютера будет на порядок дороже, чем пять обычных ПК, объединенных в кластер.
- Схема хранения: Основополагающим принципом реляционных СУБД является размещение данных в таблицах, имеющих реляционную

структуру, определяемую схемой. Реляционная модель обладает рядом замечательных формальных свойств, но многие современные приложения имеют дело с типами данных, которые плохо в эту модель укладываются. В качестве широко известных примеров упомянем текстовые документы, изображения и XML-файлы. Кроме того, большие наборы данных часто вообще не структурированы или слабо структурированы. В Hadoop в качестве основной единицы данных используется пара ключ/значение, и это достаточно гибкое решение для работы со слабо структурированными типами данных. Исходные данные в Hadoop могут быть представлены в любом формате, но в конечном итоге они преобразуются в пары ключ/значение, к которым и применяются функции обработки.

- Обработка: По сути своей SQL является высокоуровневым декларативным языком. Запрашивая данные, вы говорите, какой результат хотели бы получить, и предоставляете СУБД решать, как добиться желаемого. В парадигме MapReduce предполагается, что вы сами описываете конкретные шаги обработки данных, что в какой-то мере напоминает порождаемый СУБД план выполнения SQL-запроса. В SQL вы формулируете команды-запросы, в MapReduce пишете скрипты и программы. MapReduce допускает более общие способы обработки данных, чем SQL. Например, на основе данных можно строить сложные статистические модели или изменять формат изображений. SQL для таких задач приспособлен плохо.
- Специфика: Hadoop проектировался для автономной обработки и анализа больших объемов данных. Он не предназначен для произвольного считывания и обновления нескольких записей, то есть не может служить заменой системам оперативной обработки транзакций. На самом деле, сейчас и в обозримом будущем Hadoop лучше всего использовать для

работы с хранилищами данных, в которых запись производится однократно, а чтение многократно.

Применительно к поставленной задаче нужно учесть следующие вещи:

- Архивированные данные. В базе находятся данные собранные с 1979 по 2015 года, над которыми будет проводится статистический анализ. Сами данные будут только добавляться, но не изменяться и не обновляться.
- Большой объем данных. Совокупный объем данных только о столкновениях составляет около 10 Гб, причем данные собраны исключительно на территории Великобритании, где по сравнению, например, с Россией число транспортных средств меньше примерно в 2.5 раза.
- Масштабируемость. На сегодняшний день в базе существует только данные, вводимые непосредственно инспектором, прибывшем на место ДТП. Однако, как известно, сейчас мир постепенно движется в сторону повсеместного внедрения разнообразных датчиков и сенсоров. Например, на автодорогах уже сейчас установлены множество камер и регистраторов скорости, также разработаны датчики, встраиваемые в дорожное покрытие и регистрирующие физические и метеорологические условия. Добавление новых источников данных в сочетании с тем, что с течением времени данных скорость увеличения количества данных будет только расти, приводит к тому что систему стоит изначально разрабатывать с учетом этого.

Учитывая все ранее сказанное, было решено использовать распределенные технологии для работы с большими данными на базе Apache Hadoop.

1.2 Apache Hadoop

Hadoop — фреймворк, предназначенный для построения распределённых приложений для работы с данными очень большого объёма. Hadoop реализует вычислительную парадигму MapReduce, в которой приложение разбивается на множество независимых частей, каждая из которых может исполняться на отдельном узле. Главным языком проекта является Java.

До появления Hadoop существовал другой подход к распределенным вычислениям, который заключался в распределении работы по кластерам машин, работающих с общей файловой системой, находящейся под управлением сети хранения данных SAN. Такой подход хорош для заданий, требующих большого объема вычислений. Однако он создает проблемы, когда у узлов появляется необходимость в обращении к большим объемам данных, так как пропускная способность сети становится «узким местом», а узлы начинают простаивать. Hadoop стремится размещать данные в пределах вычислительных узлов; при этом обращения к данным выполняются быстро, так как фактически являются локальными. Эта особенность, называемая локальностью данных, лежит в основе технологии Hadoop и является причиной ее хорошей производительности.

В полностью сконфигурированном кластере под работает набор демонов, или резидентных программ на различных сетевых серверах. Каждый демон играет свою роль; некоторые запускаются только на одном сервере, другие - на нескольких. Демоны по своей функции делятся на демонов хранения и вычисления

1.2.1 Демоны хранения

Name Node. В Hadoop применяется архитектура главный/подчиненный как для распределенного хранения, так и для распределенных вычислений. Распределенная система хранения называется файловой системой Hadoop,

или HDFS (Hadoop File System). Name Node представляет собой главный демон HDFS, который распределяет низкоуровневые задачи ввода/вывода между подчиненными демонами Data Node. Name Node - это диспетчер HDFS: он ведет учет разбиению файлов на блоки, хранит информацию о том, на каких узлах эти блоки находятся, и следит за общим состоянием распределенной файловой системы.

Data Node. На любой подчиненной машине в кластере работает демон Data Node, на который возложена основная работа распределенной файловой системы - считывание и запись блоков HDFS в физические файлы, находящиеся в локальной файловой системе. Когда клиент хочет прочитать или записать HDFS-файл, демон Name Node сообщает ему, на каком узле Data Node находится каждый блок файла. Далее клиент напрямую общается с демонами Data Node, работая с локальными файлами, соответствующими блокам. Кроме того, демон Data Node может взаимодействовать с другими таким же демонами, осуществляя репликацию блоков данных для обеспечения резервирования.

Secondary Name Node. Это вспомогательный демон, который занимается мониторингом состояния кластера HDFS. От Name Node демон SNN отличается тем, что не получает и не протоколирует информацию об изменениях HDFS в режиме реального времени. Вместо этого он взаимодействует с Name Node с целью создания мгновенных снимков метаданных HDFS.

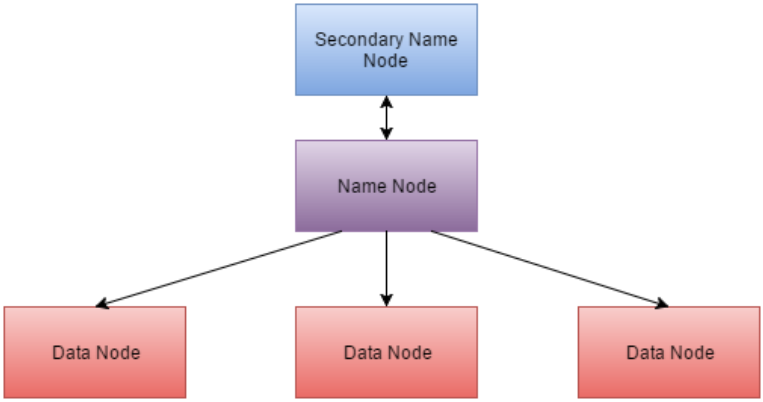

Рисунок 1. Иерархия демонов хранения Hadoop

1.2.2 Демоны вычислений

Job Tracker. Демон Job Tracker - это посредник между Hadoop и приложением. Когда вы передаете свой код кластеру, Job Tracker строит план выполнения, то есть определяет, какие файлы обрабатывать, назначает узлы различным задачам и следит за ходом исполнения этих задач. Если задача завершится неудачно, то Job Tracker автоматически перезапустит ее, возможно на другом узле; количество таких попыток определено в конфигурации кластера. В кластере Hadoop может быть только один демон Job Tracker. Обычно он работает на отдельном сервере, играющем роль главного узла кластера.

Task Tracker. Демоны вычислений, как и демоны хранения, также построены на базе архитектуры главный/подчиненный: Job Tracker - это главный демон, организующий общее выполнение задачи MapReduce, а демоны Task Tracker управляют исполнением отдельных заданий на подчиненных узлах. Это взаимодействие показано на рисунке.

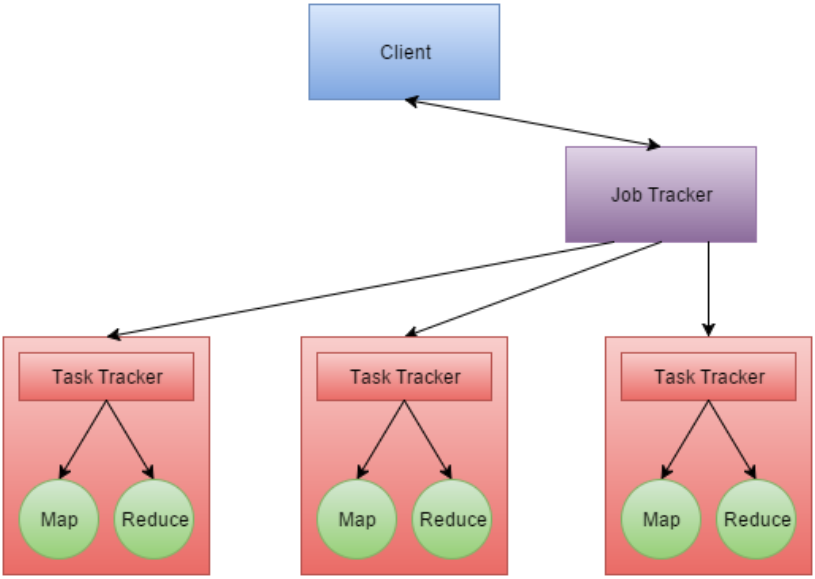

Рисунок 2. Иерархия демонов вычисления Hadoop

1.3 MapReduce

MapReduce – модель распределенной обработки данных и исполнительная среда, работающая на больших кластерах типовых машин. Работа MapReduce основана на разбиении обработки данных на две фазы: фазу отображения (map) и фазу свертки (reduce). Эти процессы создает программист и передает в среду MapReduce, которая параллельно запускает их на рабочих узлах. Различные компоненты фреймворка при этом управляют связью между узлами, заботится о деталях разбиения данных и выполнения процессов на распределенной системе и обеспечивают высокую доступность и отказоустойчивость. В ходе этого двухэтапного процесса, программа map читает данные с хранилища и выполняет содержащиеся в ней инструкции над

данными. Затем, выходные данные функции map обрабатываются инфраструктурой MapReduce перед тем, как они будут переданы функции reduce. В ходе это обработки данные сортируются и группируются. Далее результат передается процессу reduce для агрегации итоговых ответов. Иначе говоря, функция map выполняет фильтрацию и сортировку данных, а reduce – суммирование. При этом, каждый из рабочих узлов MapReduce применяет один и тот же код к своей части данных. Более подробное описание процессов map и reduce дано ниже.

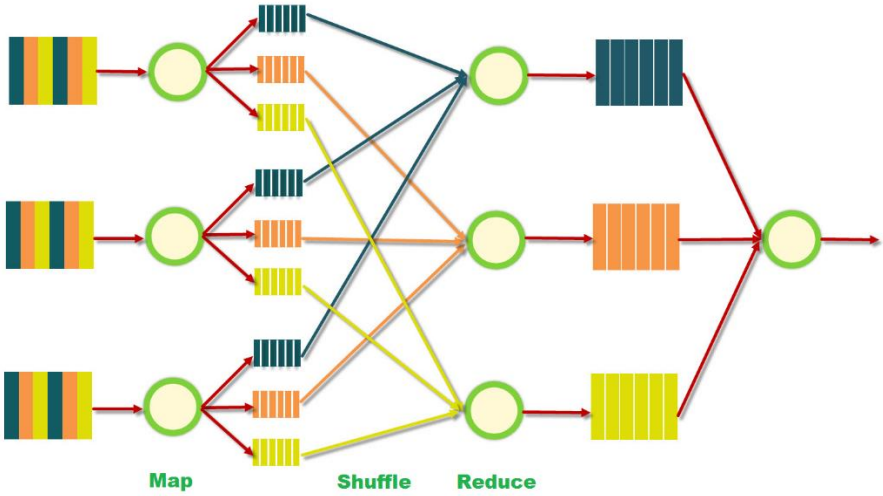

Рисунок 3. Схема работы модели MapReduce

Предложим пример в рамках поставленной задачи с данными о столкновениях транспортных средств.

Имеется входной массив данных в формате CSV, каждая запись – это набор параметров, описывающих одно ДТП: ID записи, дата, время, координаты GPS, район, серьезность аварии и другие условия:

```
<ID>, <Date>, <Time>, <Latitude>, <Longitude>, <District>
201401BS70001,09/01/2014,13:21,-0.206443,51.496345, Westminster
201401BS70002,20/01/2014,10:40,-0.216389,51.518349, Lambeth
201401BS70003,21/01/2014,08:07,-0.179731,51.497822, Westminster
201401BS70004,17/01/2014,14:11,-0.198991,51.506302, Harrow
201401BS70005,05/01/2014,18:55,-0.195118,51.487628, Harrow
```

Задача – подсчитать количество аварий в каждом районе.

- Стадия Map. На этой стадии данные предобрабатываются при помощи функции map(), которую определяет пользователь. Работа этой стадии заключается в предобработке и фильтрации данных. Работа очень похожа на операцию map в функциональных языках программирования – пользовательская функция применяется к каждой входной записи. Функция map() применяется к одной входной записи и выдаёт множество пар ключ-значение. Множество – т.е. может выдать только одну запись, может не выдать ничего, а может выдать несколько пар ключ-значение. Что будет находится в ключе и в значении – решать пользователю, но ключ – очень важная вещь, так как данные с одним ключом в будущем попадут в один экземпляр функции reduce. В рамках задачи эта функция map отфильтровывает ненужные поля и возвращает пары вида [“район”, 1]:

```
Westminster,1
Lambeth,1
Westminster,1
Harrow,1
Harrow,1
```

- Стадия Shuffle. Проходит незаметно для пользователя. В этой стадии вывод функции map «разбирается по корзинам» – каждая корзина соответствует одному ключу вывода стадии map. В

дальнейшем эти корзины послужат входом для reduce. В рамках задачи данные с выхода стадии map сгруппируются по ключу “район” и все записи с одним ключом попадут в один и тот же экземпляр функции reduce.

- Стадия Reduce. Каждая «корзина» со значениями, сформированная на стадии shuffle, попадает на вход функции reduce(). Функция reduce задаётся пользователем и вычисляет финальный результат для отдельной «корзины». Множество всех значений, возвращённых функцией reduce(), является финальным результатом MapReduce-задачи. В рамках задачи каждый экземпляр функции reduce подсчитает количество записей по каждому району, суммируя единицы в парах вида [“район”, 1] и сгенерирует итоговый результат:

```
Westminster,2
Lambeth,1
Harrow,2
```

1.4 Инструменты экосистемы Apache Hadoop

Вокруг Hadoop строилось много специализированных проектов, которые со временем превратились в целую экосистему. Самые значительные и хорошо поддерживаемые получили статус официальных подпроектов Apache Hadoop. К их числу относятся:

- Pig - высокоуровневый язык описания потоков данных;
- Hive - SQL-подобная инфраструктура для организации хранилищ данных;
- HBase - распределенная СУБД с хранением по столбцам, устроенная по образцу Google BigtaЬle;
- ZooKeeper - надежная система координации для управления состоянием, общим для нескольких распределенных приложений;

1.4.1 Pig

Pig повышает уровень абстракции при обработке больших наборов данных. MapReduce позволяет программисту задать функцию отображения, а затем функцию свертки, но для адаптации механизма обработки данных к этой схеме часто приходится использовать несколько стадий MapReduce, а это усложняет задачу. При использовании Pig структуры данных намного сложнее, с множеством значений и многоуровневой иерархией, а преобразования, применяемые к данным, намного мощнее.

Pig состоит из двух основных частей:

- Язык для описания потоков данных, называемый Pig Latin.
- Исполнительная среда для запуска программ Pig Latin. В настоящее время доступны два варианта: локальное исполнение на одной JVM и распределенное исполнение в кластере Hadoop.

Программа Pig Latin состоит из серии операций (преобразований), которые применяются к входным данным для получения выходных данных. В целом эти операции описывают поток данных, который превращается исполнительной средой Pig в исполняемое представление, а затем запускается для выполнения. Во внутренней реализации Pig трансформирует преобразования в серию заданий MapReduce, однако эта трансформация в основном остается скрытой от программиста, что позволяет ему сосредоточиться на данных, а не на природе исполнения.

Одной из целей разработки Pig была хорошая расширяемость. Настраиваются практически все стадии обработки: загрузка, хранение, фильтрация, группировка, соединение — каждая операция может быть изменена при помощи пользовательских функций (UDF, User-Defined Function). Эти функции работают с вложенной моделью данных Pig, поэтому они могут очень глубоко интегрироваться с операторами Pig. Кроме того, пользовательские функции обычно лучше подходят для повторного использования, чем библиотеки, разработанные для написания программ MapReduce.

Основной объект в Pig Latin- это «отношение». Именно с отношениями работают все операторы языка. В форме отношений представляются входные и выходные данные.

Каждое отношение представляет собой набор однотипных объектов — «кортежей» (tuples). Аналоги в БД: кортеж — это строка, отношение — это таблица. Кортежи могут в свою очередь объединятся в коллекции, называемые bag.

Кортежи соответственно состоят из нумерованных или именованных объектов — «полей», произвольных базовых типов (число, строка, булева переменная и т.д.).

1.4.2 Hive

Hive - это пакет для организации хранилищ данных, построенный на базе Hadoop. Ориентирован он на аналитиков, которые уверенно владеют SQL и нуждаются в средстве для выполнения произвольных запросов, агрегирования и анализа данных, объем которых таков, что требуется Hadoop. Для взаимодействия с Hive применяется SQL-подобный язык запросов, называемый HiveQL.

Hive представляет из себя движок, который превращает SQL-запросы в цепочки map-reduce задач. Движок включает в себя такие компоненты, как Parser(разбирает входящие SQL-запросы), Optimizer(оптимизирует запрос для достижения большей эффективности), Planner (планирует задачи на выполнение) Executor(запускает задачи на фреймворке MapReduce.

Для работы Hive также необходимо хранилище метаданных. Дело в том что SQL предполагает работу с такими объектами как база данных, таблица, колонки, строчки, ячейки и т.д. Поскольку сами данные, которые использует Hive хранятся просто в виде файлов на HDFS — необходимо где-то хранить соответствие между объектами Hive и реальными файлами. Обычно метахранилище размещается в реляционной базе данных.

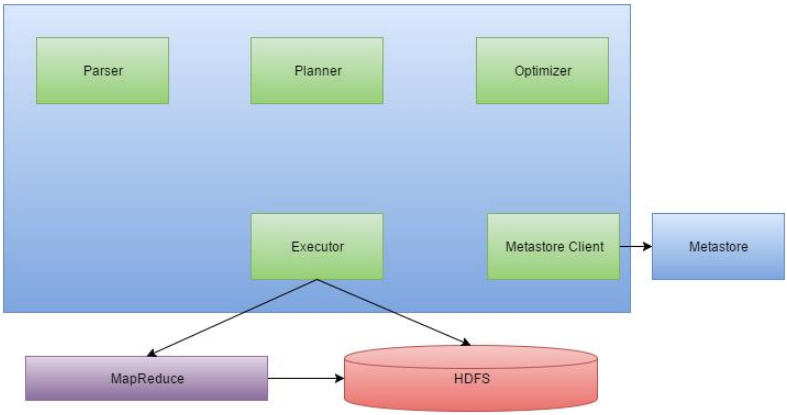

Рисунок 4. Структурная схема Apache Hive

Pig и Hive обладают приблизительно равными возможностями и созданы для одной и той же цели – абстрагировать разработчика от непосредственной разработки программ в рамках MapReduce. Но тем не менее несколько отличий существует.

- Pig реализует процедурный подход, в то время как Hive использует декларативный SQL подход. Поэтому при использовании сложных выборок запросы в HQL становятся очень громоздкими, а Pig позволяет разбивать логику на блоки, каждый шаг можно развернуто описывать комментариями.
- Разработка на Pig более сложна и предполагает изучение Pig Latin, в то время как HQL практически идентичен всем знакомому SQL.

Так как развитие системы предполагает увеличение источников данных, и сложность запросов будет только расти, было решено изначально использовать Apache Pig.

1.4.3 HBase

HBase — распределенная столбцово-ориентированная база данных, построенная на основе HDFS. Это приложение Hadoop, используемое в ситуациях, когда вам необходимо организовать произвольный доступ для чтения/записи к очень большим наборам данных в реальном времени.

Приложения хранят данные в таблицах, состоящих из строк и столбцов. Для ячеек таблицы (пересечения строк и столбцов) действует контроль версии. По умолчанию в качестве версии используется временная метка, автоматически назначаемая HBase на момент вставки. Содержимое ячейки

представляет собой неинтерпретируемый массив байтов. Ключи строк таблицы тоже являются байтовыми массивами, поэтому теоретически ключом строки может быть что угодно — от строк до двоичных представлений long и даже сериализованных структур данных. Строки таблицы сортируются по ключу строк (первичному ключу таблицы). Сортировка осуществляется в порядке следования байтов. Все обращения к таблице выполняются по первичному ключу.

Колонки организованны в группы колонок, называемые Column Family. Как правило в одну Column Family объединяют колонки, для которых одинаковы паттерн использования и хранения. Записи физически хранятся в отсортированном по RowKey порядке. При этом данные соответствующие разным Column Family хранятся отдельно, что позволяет при необходимости читать данные только из нужного семейства колонок.

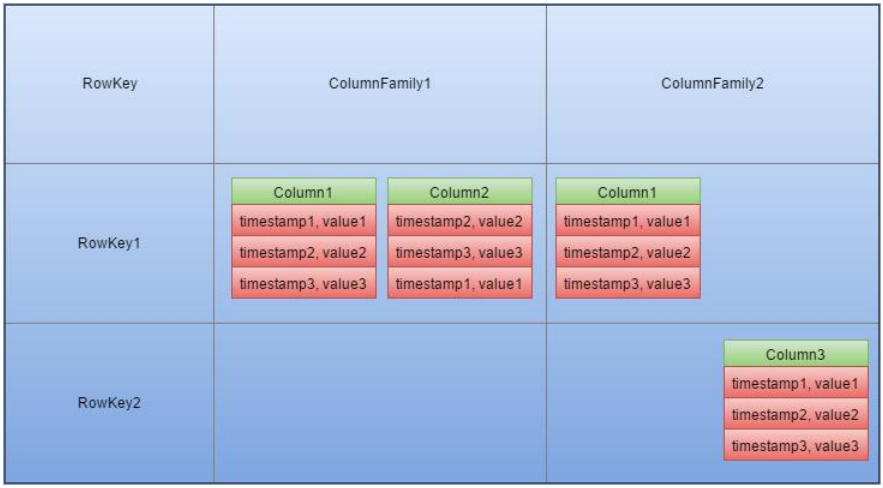

Рисунок 5. Схема организации данных в HBase

HBase является распределенной базой данных, которая может работать на десятках и сотнях физических серверов, обеспечивая бесперебойную работу

даже при выходе из строя некоторых из них. Поэтому архитектура HBase довольна сложна по сравнению с классическими реляционными базами данных.

HBase для своей работы использует две основных сущности:

- Region Server — обслуживает один или несколько диапазонов записей соответствующих определенному диапазону ключей (регионов). Каждый регион содержит:
  - Основное хранилище данных. HBase работает поверх HDFS, причем использует собственный формат файлов – HFile, в котором данные хранятся в отсортированном по ключу в лексиграфическом порядке.
  - Буфер на запись. Так как данные хранятся в HFile в отсортированном порядке — обновлять эти файлы на каждую запись довольно дорого, поэтому данные при записи попадают в специальную область памяти, где накапливаются, и через некоторое время записываются в основное хранилище.
  - Кэш на чтение. Позволяет существенно экономить время на данных которые читаются часто.
  - При использовании буфера на запись существует некоторый риск потери данных из-за сбоя, поэтому, для обеспечения отказоустойчивости, все операции перед исполнением попадают в специальный журнальный файл – так называемый Write Ahead Log.
- Master Server — главный сервер в кластере HBase. Master управляет распределением регионов, ведет их реестр, управляет запусками регулярных задач и делает другую полезную работу.

Для координации действий между сервисами HBase использует Apache ZooKeeper, специальный сервис, предназначенный для управления конфигурациями и синхронизацией сервисов.

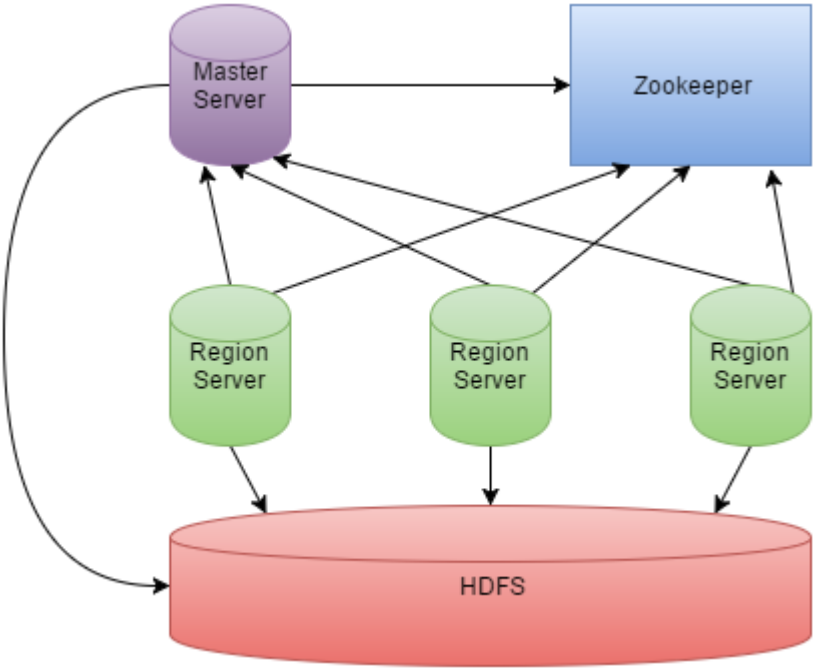

Рисунок 6. Функциональная схема построения кластера HBase

При увеличении количества данных в регионе и достижении им определенного размера HBase запускает split, операцию разбивающую регион на 2. Для того чтобы избежать постоянных делений регионов — можно заранее задать границы регионов и увеличить их максимальный размер.

HBase предназначен главным образом для произвольного чтения и доступа к большим данным. Использование HBase оправдано, когда данные часто обновляются и удаляются и нужен произвольный доступ к данным по определенным ключам. В силу этих причин в рамках нашей задачи Apache HBase использоваться не будет.

1.5 Инструменты визуализации данных

Для визуализации результатов были выбраны инструменты, разработанные компанией Elastic и работающие в тандеме – Elasticseach и Kibana. Эти инструменты являются распределенными и легко интегрируется с элементами экосистемы Hadoop.

1.5.1 Elasticsearch

Elasticsearch – это поисковый сервер, разработанный на базе свободной библиотеки высокоскоростного полнотекстового поиска Apache Lucene. Он предназначен для индексации и поиска любого вида данных, в том числе в реальном времени и может работать и как поисковый движок и как хранилище данных. Elasticsearch может быть установлен как на одну машину, так и на кластер, то есть является распределённой системой, что позволяет ему работать с любыми объемами данных.

Основными понятиями, которыми оперирует Elasticsearch являются:

- Индекс – основная сущность, в которой хранятся данные. По сути своей является таблицей с записями любого вида, которые соответствующим образом приготовлены для быстрого поиска. Индекс идентифицируется по имени (которое должно быть в нижнем регистре) и это имя используется как ссылка на этот индекс при выполнении операций индексирования, поиска, обновления и удаления документов, содержащихся в индексе. В одном кластере, можно создать произвольное количество индексов.
- Документ – основная сущность, которая хранится в Elasticsearch. По сути является строкой в таблице – индексе.
- Тип документа – так как каждый индекс может хранить документы разной структуры, каждый индекс делится на типы, каждый из которых

имеет свои фиксированные поля. Внутри каждого типа документы должны иметь одинаковый формат и структуру полей внутри себя. Внутри индекса, можно определить один или несколько типов

- Шард - Индекс может потенциально хранить большое количество данных, которые могут превышать аппаратные ограничения на одном узле. Например, один индекс одного миллиарда документов может занимать свыше 1TB места на диске, что может не поместится на диск одного узла или может быть слишком медленным для обслуживания поисковых запросов на одном узле. Чтобы решить эту проблему, Elasticsearch предоставляет возможность разделения вашего индекса на несколько частей, которые называются шардам. Каждый шард является самодостаточным и полнофункциональным независимым «индексом», которые может размещаться на любом узле кластера. Шардинг является важным по двум основным причинам:
    - Он позволяет горизонтально разбивать/масштабировать тома данных
    - Он позволяет распространять и параллелизировать операции с шардингами (потенциально на нескольких узлах) что увеличивает производительность/пропускную способность

Механика работы распространения шарда а также агрегации находящихся в нём документов в результаты поиска, полностью управляется Elasticsearch и прозрачна для пользователя.

1.5.2 Kibana

Kibana – плагин для Elasticsearch с открытым исходным кодом, предназначенный для визуализации данных. Kibana работает поверх сервера Elasticsearch и по сути является веб-интерфейсом для отображения данных, находящихся в индексах Elasticsearch.

Поддерживает различные виды двухмерных диаграмм (круговая, линейная, областная, столбчатая), а также картограммы и способен осуществлять простейшие группировки данных по полям и арифметические операции такие как подсчет, суммирование, нахождение среднего значения.

1.6 Инструменты обработки данных в реальном времени

1.6.1 Apache Storm

Для прогнозирования количества ДТП в реальном времени был выбран фреймворк с открытым исходным кодом Apache Storm. Storm это распределенная система, направленная на обработку больших потоков данных, аналогичная Apache Hadoop, но работающая в реальном времени.

Ключевыми особенностями Storm являются:

- Масштабируемость. Задачи обработки распределяются по узлам кластера и потокам на каждом узле.
- Гарантированная защита от потери данных.
- Простота развертывания и сопровождения.
- Восстановление после сбоев. Если какой-либо из обработчиков отказывает, задачи переадресуются на другие обработчики.

Приложение Storm проектируется как «топология» в виде направленного ациклического графа (DAG), вершинами которого являются объекты, называемые Bolt и Spout.

Spout – это объект (класс) который отвечает за прием данных из внешнего источника и дальнейшую пересылку их по топологии.

Bolt – это объект (класс) который обрабатывает данные, поступившие в него.

Топология может состоять из нескольких Bolt и Spout, каждый из которых может быть соединен с одним или несколькими другими.

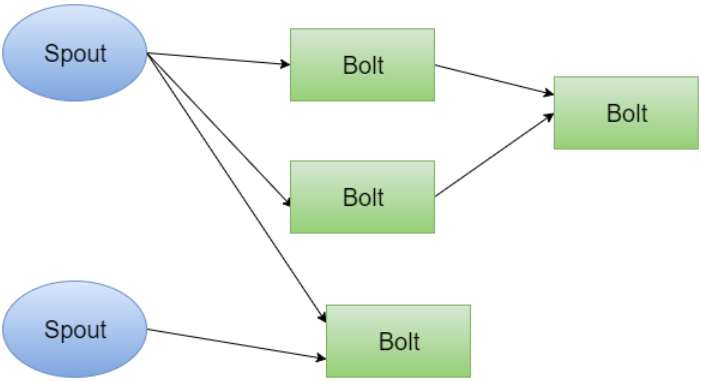

Рисунок 7. Пример топологии Apache Storm

Как и Apache Hadoop, в Apache Storm существуют демоны, отвечающие за управление служебными процессами:

- Nimbus – работает на главном узле и отвечает за распределение кода по кластеру, присваивание задач рабочим узлам и мониторингом неисправностей
- Supervisor – работает на рабочем узле и отвечает за исполнение части топологии.

Управление кластером, производится службой координации Zookeeper.

1.6.2 Apache Kafka

Apache Kafka – распределенный брокер сообщений работающий по принципу «публикация-подписка» с открытым исходным кодом.

Ключевые особенности Kafka:

- Распределённая система, которую легко масштабировать,
- Поддержка высокой пропускной способности как со стороны источников, так и для систем-подписчиков,
- Поддержка объединение подписчиков в группы,
- Возможность временного хранения данных для последующей пакетной обработки (например, для ETL-процессов)

Основные компоненты архитектуры:

- Topic - Поток сообщений определенного типа, категория, в соответствии с которой публикуется то или иное сообщение.
- Producer – процесс, публикующий сообщения в тему
- Broker – один сервер из кластера Kafka
- Consumer – процесс, который читает сообщения из темы. Может подписаться на одну или несколько тем и использовать сообщения, забирая данные от брокеров.
- Partition – часть темы, упорядоченная, неизменная последовательность сообщений. Внутри каждого partition сообщения идентифицируются уникальным номером – offset.

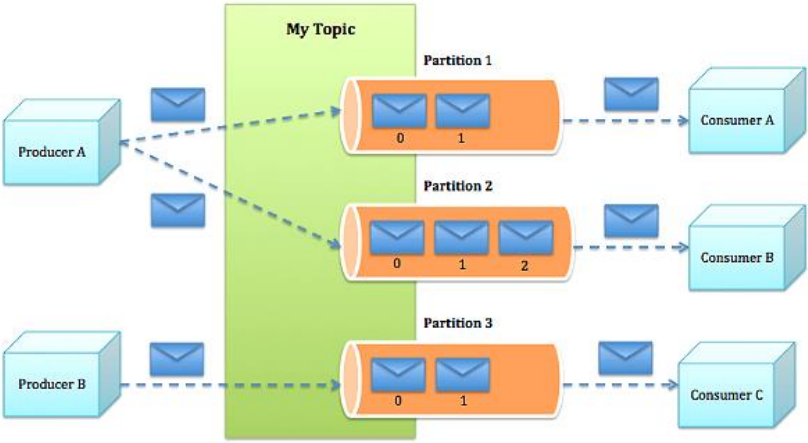

Рисунок 8. Функциональная схема Apache Kafka

1.6.3 Apache Flume

Так как для каждого источника данных Apache Kafka требует собственный producer, который будет отправлять сообщения в темы, для легкого чтения данных из разных источников был выбран инструмент Apache Flume. Flume предназначен для того, чтобы управлять потоками данных: собирать их из различных источников и направлять их различным потребителям. Является распределенной системой с высокой надежностью и отказоустойчивостью.

Flume изначально поддерживает множество источников и потребителей данных: Kafka, HDFS, локальные файлы, HTTP и т.д., а также обладает очень прост в использовании. Каждый исполняемый процесс Flume (agent) конфигурируется файлом конфигурации, в котором описываются вид источника, канала (хранилище где хранятся сообщения пока они не запишутся в приемник) и потребителя.

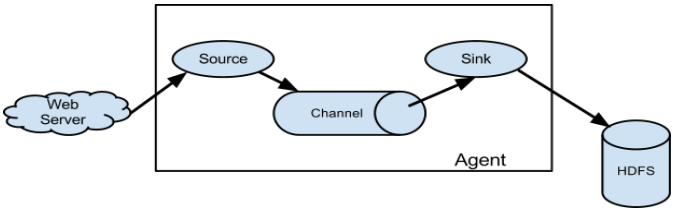

Рисунок 9. Пример агента Flume

1.7 Язык R

Для реализации прогнозирования предварительно необходимо построить регрессионную модель по имеющимся данным. Среди множества инструментов был выбран язык R.

R — язык программирования для статистической обработки данных и работы с графикой, а также программная среда с открытым исходным кодом, развиваемая в рамках проекта GNU. Сегодня R является безусловным лидером среди свободно распространяемых систем статистического анализа. В среде R реализованы многие статистические методы: линейные и нелинейным модели, проверка статистических гипотез, анализ временных рядов, классификация, кластеризация, графическая визуализация. Язык R позволяет определять собственные функции. Многие функции R написаны на самом R.

Преимущества R:

- Бесплатное распространение
- Создан специально для статистического анализа
- Способен работать с большими объемами данных
- Может быть дополнен дополнительными библиотеками
- Возможность написания скриптов, которые потом загружаются в R и интерпретируются им

1.8 Используемые методы

1.8.1 Метод обратных взвешенных расстояний

Так как количество погодных станций ограничено, а ДТП случаются по всей рассматриваемой территории, то чтобы определить значений погодных характеристик, а именно температуры и количества осадков в конкретной географической точке (точке столкновения) необходимо использовать какой-либо метод интерполяции.

Из всего многообразия интерполяционных методов было решено воспользоваться методом обратных взвешенных расстояний, так как он лучше всего удовлетворяет условиям и исходным данным задачи.

Суть данного метода заключается в том, что точки, находящиеся ближе к тем, в которых производится оценивание, оказывают большее влияние, по сравнению с удалёнными точками. Для более точного определения описания топографии набор точек, по которым будет осуществляться интерполяция, выбираются в некоторой окрестности определяемой точки, так как они оказывают наибольшее влияние на значение характеристики в ней. Это производится таким образом. Выбирается максимальный радиус поиска или количество точек, ближайших по абсолютному расстоянию от определяемой точки. После этого задается вес значению выбранной характеристики в каждой выбранной точке, который вычисляется в зависимости от расстояния до определяемой точки. Таким образом достигается внос большего вклада более близких точек в определение интерполируемой высоты по сравнению с более удаленными точками.

Значение в конкретной точке определяется по формуле

$$\hat{z}(s_0) = \sum_{i=1}^{N} \alpha_i z(s_i)$$

Где $\hat{z}$ – это искомое значение для точки s_0, $z(s_i)$ – измеренное значение в точке s_i, N - число опорных точек, находящихся в окрестности искомой точки и используемых в вычислениях, α_i - веса, присвоенные каждой опорной точке, из числа тех, ко-торые будут использованы в вычислениях и определяющиеся по формуле

$$\alpha_i = d_i^{-p} / \sum_{j=1}^{N} d_j^{-p}$$

Где α_i – это искомый весовой коэффициент для точки s_i, d_i – это расстояние между искомой точкой s_0, и опорной точкой s_i, p – степень метода.

Степень метода определяет скорость уменьшения весов по мере отдаления от рассматриваемой точки. По мере увеличения значения p, веса отдаленных точек будут стремительно уменьшаться.

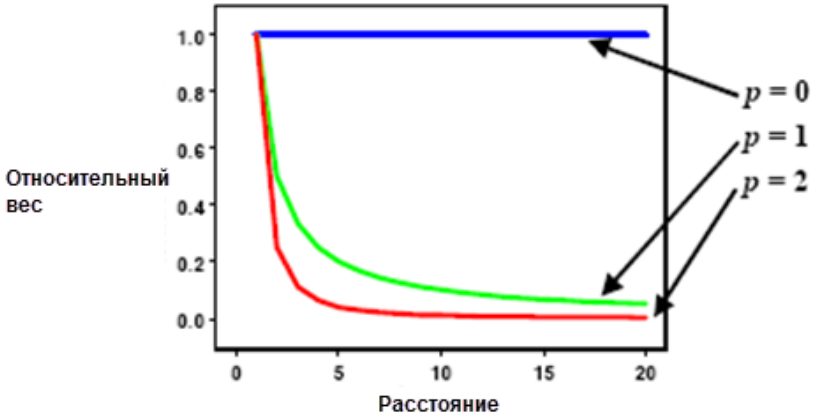

Рисунок 10. Зависимость относительного веса опорных точек от расстояния при разных степенях метода

Иллюстрация уменьшения весов с изменением расстояния

Применительно к нашей задаче, было решено рассматривать три ближайшие погодные станции и взять степень метода $p = 1$.

1.8.2 Линейная регрессия

Для прогнозирования количества аварий была выбрана линейная регрессионная модель, как наиболее простая и удобная.

Линейная регрессия – используемая в статистике регрессионная модель зависимости одной (зависимой) переменной от другой или нескольких других (независимых) переменных с линейной функцией зависимости. Линейная регрессия сводится к нахождению параметров уравнения вида:

$$y = b_1 x + b_0$$

Это уравнение позволяет по заданным фактическим значениям фактора иметь теоретические значения результативного признака. На графике

теоретические значения представляют собой линию регрессии, а фактические – разброс точек в виде поля корреляции (рис. 1).

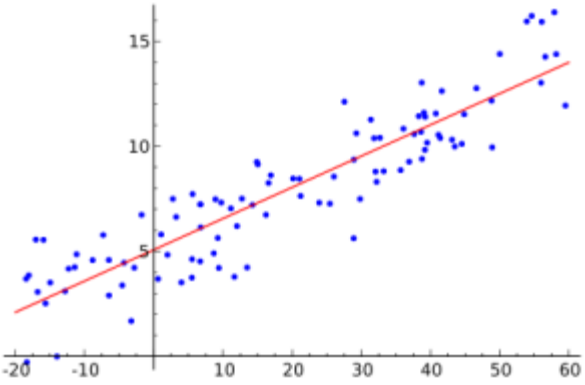

Рисунок 11. Графическое представление линии регрессии и поля корреляции

Построение линейной регрессии сводится к оценке ее параметров – b_1 и b_0.

Оценки параметров линейной регрессии могут быть найдены разными методами. Классический же подход к оцениванию параметров линейной регрессии основан на методе наименьших квадратов (МНК). МНК позволяет получить такие оценки параметров b_1 и b_0, при которых сумма квадратов

отклонений фактических значений результативного признака от расчетных минимальна:

$$\sum_i (y_i - y_{x_i}) \to min$$

Для решения задачи минимизации вводится функция невязки, составляется условие минимума функции невязки, являющееся системой линейных уравнений, которая решается любым удобным способом.

Для получения наилучших оценок необходимо условий Гаусса — Маркова.

В языке R в качестве оценки качества построенной модели используется коэффициент детерминации. Коэффициент – это доля дисперсии зависимой переменной, объясняемая рассматриваемой моделью зависимости, то есть объясняющими переменными. Или иначе – это единица минус доля необъяснённой дисперсии (дисперсии случайной ошибки модели, или условной по факторам дисперсии зависимой переменной) в дисперсии зависимой переменной:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

где y_i – значения наблюдаемой переменной, $\bar{y}$– среднее значение по наблюдаемым данным, $\widehat{y_i}$– модельные значения, построенные по оцененным параметрам.

Коэффициент детерминации принимает значения от 0 до 1. Чем ближе значение коэффициента к 1, тем качественнее. Для приемлемых моделей предполагается, что коэффициент детерминации должен быть хотя бы не меньше 50 % (в этом случае коэффициент множественной корреляции превышает по модулю 70 %). Модели с коэффициентом детерминации выше 80 % можно признать достаточно хорошими (коэффициент корреляции превышает 90 %). Значение коэффициента детерминации 1 означает функциональную зависимость между переменными.

2. Реализация

2.1 Требования

Разрабатываемая система должна удовлетворять следующим требованиям:

- Легкая горизонтальная масштабируемость
- Высокая отказоустойчивость
- Высокое быстродействие
- Легкое последующее расширение на разные источники данных
- Реализация следующего функционала:
  - Отображение данных об авариях на карте
  - Отображение количества аварий на динамически задаваемом временном промежутке.
  - Вычисление статистической информации с последующей визуализацией.

Примеры use cases:

- Соотношение количества аварий по дням недели
- Соотношение количества аварий по серьезности аварий
- Соотношение количества аварий по количеству жертв
- Соотношение марок автомобилей по количеству аварий
- Соотношение количества аварий в зависимости от погодных условий.
- Группировка водителей транспортных средств, попавших в ДТП по социальным признакам: пол, возраст, район проживания.
- Прогнозирование количества аварий по районам на основе информации о плотности транспортного потока в реальном времени и

визуализация предупреждений при высоком спрогнозированном значении.

2.2 Архитектура проекта

Архитектура проекта будет построена на основе лямбда архитектуры.

Лямбда архитектура — это архитектура обработки данных, созданная для анализа больших объемов данных, посредством использования преимуществ пакетной и потоковой обработки данных. Этот подход использует пакетную обработку для точного и комплексного анализа архивированных исторических данных, в то время как потоковая обработка предоставляет результаты обработки онлайн данных в реальном времени, что позволяет увеличить пропускную способность, быстродействие и отказоустойчивость.

Лямбда-архитектура имеет структуру, состоящую из трех уровней:

- Пакетный уровень (batch layer) – архив сырых исторических данных. Чаще всего, это «озеро данных» на базе Hadoop. Этот уровень поддерживает и оперирует пакетной передачей данных. Важно заметить, что старые данные здесь остаются неизменными – происходит лишь добавление новых.
- Уровень ускорения (speed layer) отвечает за обработку данных, поступающих в систему в реальном времени. Представляет собой совокупность складов данных, в которых те находятся в режиме очереди, в потоковом или в рабочем режиме.
- Сервисный уровень (serving layer) выходные данные из пакетного и потокового уровня хранятся на сервисном уровне, который отвечает за индексацию (для быстрого выполнения запросов) и обработку результатов вычислений двух других уровней.

Проект будет построен по лямбда архитектуре по следующим причинам:

- Большие объемы данных
- Наличие как архивных данных, так и данных поступающих в режиме реального времени
- Высокие требования по быстродействию, масштабируемости и отказоустойчивости.

2.2.1 Пакетный уровень:

В данном проекте на пакетном уровне будет использоваться Apache Hadoop и будет происходить предварительная фильтрация, обработка и объединение архивных данных из разных источников. Результаты будут использоваться на сервисном уровне для формирования простейших запросов к ним, а также на уровне ускорения.

Элементы пакетного уровня:

HDFS: В файловой системе Hadoop будут хранятся архивированные данные о столкновениях, плотности траффике и погоде. Также, в нее будут сохранятся результаты работы Pig скриптов.

Pig: Фреймворк Pig выполняет над данными, хранящимися в HDFS необходимую пакетную обработку посредством MapReduce задач. Это включает в себя реализацию некоторых use cases, а также обработку данных для построения регрессионной модели.

R: Язык R выполняет построение линейной регрессионной модели на основе архивных данных, предварительно обработанных Pig скриптом.

2.2.2 Потоковый уровень

На уровне ускорения будет происходить real-time обработка данных на основе результатов, полученных при пакетной обработке. Моделью, построенной по результатам работы пакетного уровня, будет производится

прогнозирование количества аварий на основе данных о плотности потока, которые поступают в реальном времени.

Элементы потокового уровня:

Flume: Flume используется в качестве коннектора между непосредственным источником данных и исполнительной частью проекта. В силу своей универсальности может брать данные из любого источника.

Kafka: брокер сообщений Kafka используется для распределения сообщений между топологиями Apache Storm.

Storm: фреймворк Apache Storm используя построенную модель и данные поступающие из Kafka в режиме реального времени прогнозирует количество аварий по каждому району.

2.2.3 Сервисный уровень

На сервисном уровне данные обработанные на пакетном и потоковом уровне индексируются для быстрого поиска и эффективной реализации

Элементы сервисного уровня:

Elasticsearch; Elasticsearch используется в качестве системы хранения части данных, к которым нужно проводить быстрые запросы или визуализировать.

Kibana: Kibana используется для визуализации данных в том числе с использованием простейшей группировки и фильтрации, а также слежения за метриками, вычисленными Apache Storm в реальном времени.

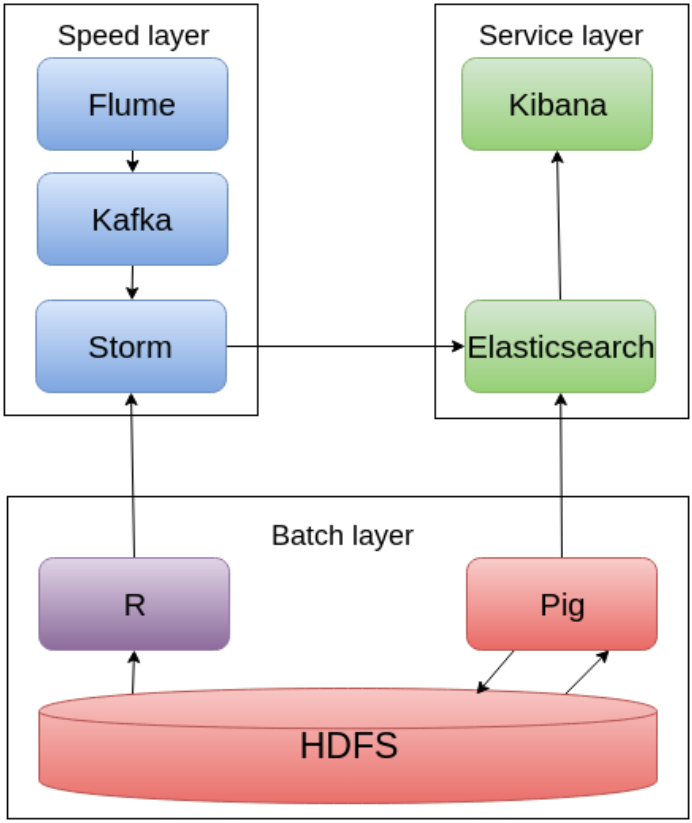

Рисунок 12. Архитектура системы

2.3 Разработка необходимого функционала

Все разработанные use cases можно разделить на три категории

- Пакетная обработка данных из одного источника
- Пакетная обработка данных из нескольких источников
- Потоковая обработка данных

Далее будут более подробно описаны методы реализации каждой из трех категорий

2.3.1 Пакетная обработка данных из одного источника

Задачи этой категории заключаются в получении статистики, посредством группировки записей из таблицы о столкновениях по одному или нескольким полям и подсчет количества вхождений в каждую группу. Реализован двумя способами: обработка непосредственно Pig скриптами и загрузка данных Elasticsearch и последующая визуализация в Kibana.

Алгоритм работы Pig скриптов является идентичным для группировки по любым полям:

1) Загрузка данных из HDFS. В языке Pig выполняется стандартной функцией.
2) Группировка по интересующим полям. В синтаксисе языка Pig это производится функция GROUP BY
3) Подсчет необходимых метрик. В языке Pig производится функциями COUNT, AVG, SUM и другими.
4) Запись данных в HDFS, в локальный файл или вывод на консоль.

Для загрузки данных из HDFS в Elasticsearch также используется Pig. В этом случае не нужно производить группировку и подсчет, так как эти возможности реализованы в Kibana, которая при визуализации может сгруппировать данные и вычислить метрики. Для того чтобы эффективно использовать Kibana, а именно иметь возможность строить диаграммы в динамическом диапазоне времени, а также наносить точки на картограммы, необходимо совершить предварительную обработку форматов полей

координат и даты и соответственно сконфигурировать нужный индекс Elasticsearch.

Дату необходимо привести в формат Unix timestamp, то есть целое число, показывающее количество секунд, прошедших с 00:00 1 Января 1970 года.

Координаты необходимо привести в вид строки в формате JSON со следующей структурой:

```
location: {
    lon: <значение долготы>,
    lat: <значение широты>
}
```

Конфигурация индекса Elasticsearch заключается в передаче на сервер файла в формате JSON со следующей структурой:

```
{
  "settings": {
    "number_of_shards": <количество шардов>,
    "number_of_replicas" : <количество реплик>
  },

      "mappings": {
        "<название типа>" : {
          "properties" : {
            "location" : {
       "type" : "geo_point"
     }
          },
         "timestamp" : {
       "type" : "date",
              "format" : "epoch_second"
          }
       }
     }
}
```

Elasticsearch, а следовательно и Kibana может интерпретировать значения времени и координат и работать с ними соответствующе только в таких форматах. Остальные типы данных, такие как integer, double или string определяются автоматически при записи данных.

2.3.2 Пакетная обработка данных из нескольких источников

В этой категории присутствуют две задачи:

1) Объединение данных о столкновениях с данными о погоде и с данными о траффике для сбора статистики.
2) Объединение данных о столкновениях с данными о траффике для создания модели прогнозирования и построение этой модели.

Суть первой задачи состоит в приведении каждой записи о столкновении погоду и приблизительное значение плотности транспортного потока в этот момент в этой точке для последующей аналитики. Эта задача будет реализована с помощью Pig скрипта. Алгоритм его работы приведен на рисунке.

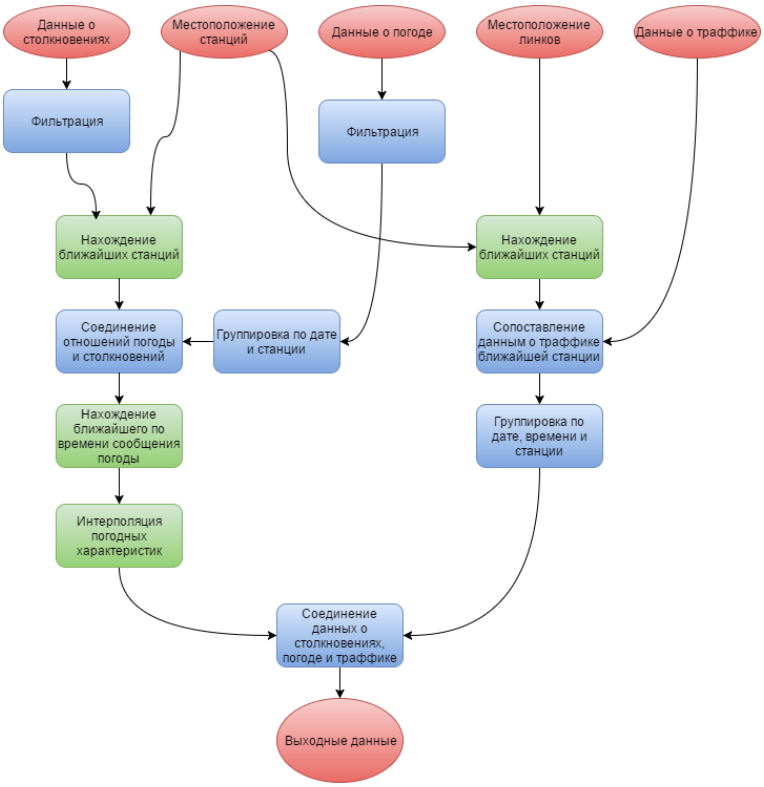

Рисунок 13. Алгоритм работы Pig скрипта

Смысл алгоритма в том, что каждому столкновению ставится в соответствие три ближайших станции, значения погодных характеристик которых интерполируются методом взвешенных обратных расстояний.

Также для ближайшей станции вычисляется среднее значение плотности траффика за необходимый час, посредством определения ближайшей станции для каждого линка (участка дороги на котором ведется наблюдение за траффиком) и вычисления среднего значения плотности траффика в районе данной станции. Далее все данные объединяются в единое отношение, каждая запись в котором содержит параметры одного ДТП, интерполированные

значения количества осадков и температуры и среднее значение плотности траффика в этом районе.

На рисунке синим цветом обозначены шаги, которые реализованы с помощью стандартных функций и операторов языка Pig, а зеленым цветом обозначены UDF.

В этом алгоритме используются три UDF:

1) Нахождение ближайших станций. Эта функция принимает на вход данные о местоположении станции, где каждому номеру станции соответствует два значения – широта и долгота, а также либо данные о столкновениях (в этом случае определяются три ближайшие станции), либо данные о плотности траффика (в этом случае определяется одна ближайшая к данному линку). Нахождение расстояний производится по формуле нахождения расстояния между двумя точками на поверхности шара:

$$D = R \arccos(\sin\varphi_1 \sin\varphi_2 + \cos\varphi_1 \cos\varphi_2 \cos(\lambda_1 - \lambda_2))$$

Где R – радиус земли, φ_1 и λ_1 - широта и долгота первой точки, φ_2 и λ_2 - широта и долгота второй точки

2) Нахождение ближайшего времени. Данные о погоде организованы в виде сообщений для каждой станции, в которых находится информация о погоде за прошедший час. Чтобы обеспечить максимальную точность необходимо сопоставить каждому ДТП то сообщение погоды, которое является наиболее близким к нему по времени. Для этого перед исполнением UDF сообщения группируются по дате и номеру ДТП, таким образом для каждой аварии формируется bag со всеми сообщениями погоды от нужной станции за эту дату. Далее UDF находит среди всех сообщений в этом bag ближайшее по времени.

3) Интерполяция погодных характеристик. Для достижения лучшей точности погодные характеристики (количество осадков, температура) для каждого ДТП интерполируются по трем ближайшим станциям методом обратных взвешенных расстояний.

Суть второй задачи состоит в приготовления данных для построения модели прогнозирования. Так как цель модели – прогнозировать количество аварий на основании данных о количестве автомобилей, то и данные для построения модели должны быть в виде таблицы где каждая строка – это количество аварий и плотность траффика, сгруппированные по какому-то признаку. Очевидно, что главным параметром, влияющим на плотность траффика, является время. В утренние и вечерние «часы пик» машин на дорогах гораздо больше чем ночью. В соответствии с этим работает и алгоритм для получения исходных данных для модели.

Данные о столкновениях группируются по дате и часу и для этой группы происходит подсчет количества ДТП. Данные о траффике также группируются по дате и часу и вычисляется сумма траффика для этой группы. После чего данные объединяются и в результате для каждого часа каждого дня имеются данные о количестве аварий и траффике.

На основе этого с помощью средств языка R строится линейная регрессионная модель методом наименьших квадратов, то есть определяются коэффициенты b_1и b_0 в модели

$$y = b_1 x + b_0$$

Где y – прогнозируемое количество аварий, x – значение плотности траффика. Также R определяет параметры точности получившейся модели.

2.3.3 Потоковая обработка данных

Задача этой категории – принять данные поступающие в реальном времени и с помощью построенной модели вычислить прогнозируемые значения аварий для разных территориальных областей. В качестве опорных пунктов для разбиения на области были выбраны те же погодные станции системы MIDAS, которые были использованы при обработке погодных данных.

Прием данных осуществляется Apache Flume, который затем передает данные в брокер Apache Kafka. В тестовом режиме Flume читает данные из локальной директории. Вычисления производятся приложением Apache Storm. Кроме того, что такая архитектура обладает высокой пропускной способностью, она может быть легко расширена на другие источники и виды данных.

Сама топология Storm организована в виде одного Spout и трех Bolt. Как только во входной теме Kafka появляется новое сообщение Kafka Spout читает его и передает объекту Flow Bolt, который производит вычисления в режиме реального времени. Вычисления сводятся к тому, что при приходе каждого нового сообщения этот объект вычисляет суммарный траффик за предшествующие 60 минут, вычисляет по нему прогнозируемое значение количества аварий и обновляет списки, в которых хранятся данные за последний час. После чего он выдает результат – сообщение которое содержит дату, время, плотность траффика и расчетное количество аварий оставшимся двум объектам. Также в процессе вычислений Flow Bolt определяет уровень аварийной опасности в районе. Если расчетное количество аварий превышает 1.4 уровень считается условно «желтым», если превышает 2.0 уровень считается условно «красным». ES Sink записывает результаты в Elasticsearch для последующей визуализации в Kibana. Kafka Sink записывает результаты в выходную тему Kafka, для возможности вывода результатов в любой другой удобный ресурс.

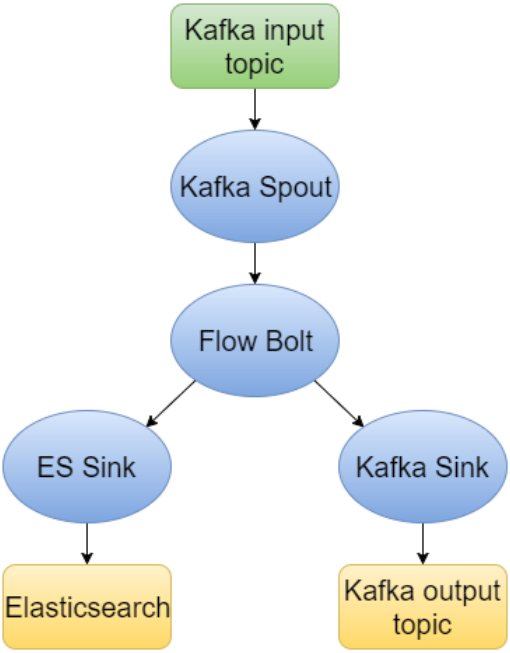

Рисунок 14. Структурная схема топологии Storm

3. Результаты

3.1 Статистика

3.1.1 Отображение данных об авариях на карте

Результат представлен в веб-интерфейсе Kibana в виде тепловой карты. Это позволяет эффективно отследить аварийность городов, районов и перекрестков

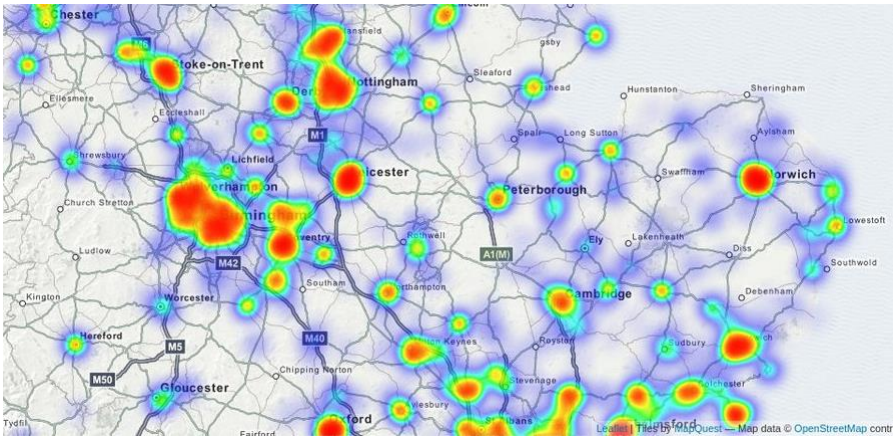

Рисунок 15. Отображение аварий на карте. Уровень городов.

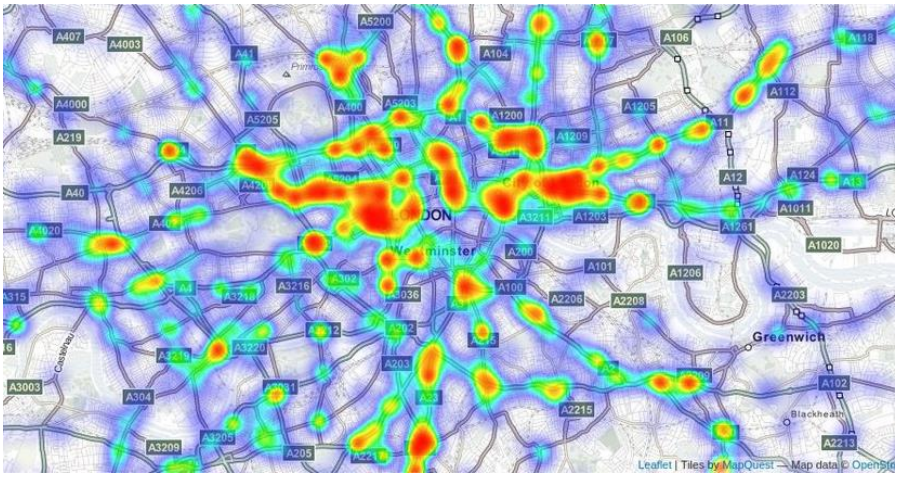

Рисунок 16. Отображение аварий на карте. Уровень районов.

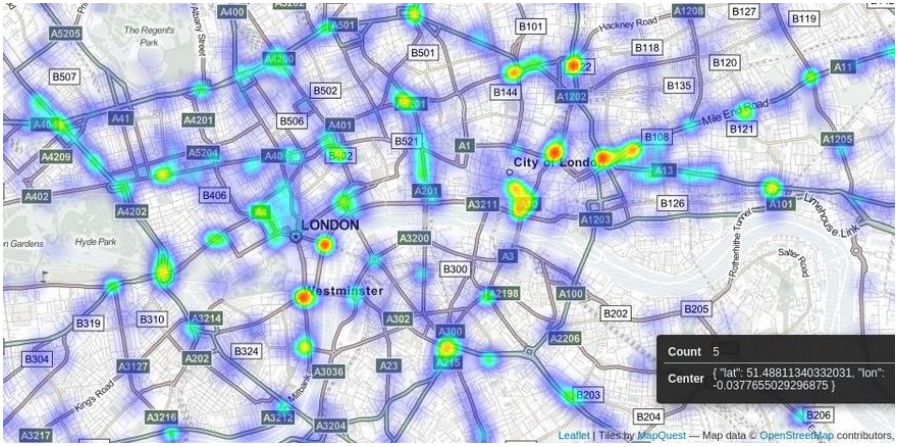

Рисунок 17. Отображение аварий на карте. Уровень перекрестков.

3.1.2 Отображение количества аварий на временном промежутке

В случае если в индексе есть поле с типом date, Kibana позволяет отображать данные на временном промежутке с задаваемым интервалом и шагом. Это позволяет отслеживать максимумы, минимумы и другие

характерные свойства динамики количества аварий за любое время – за год, месяц, день.

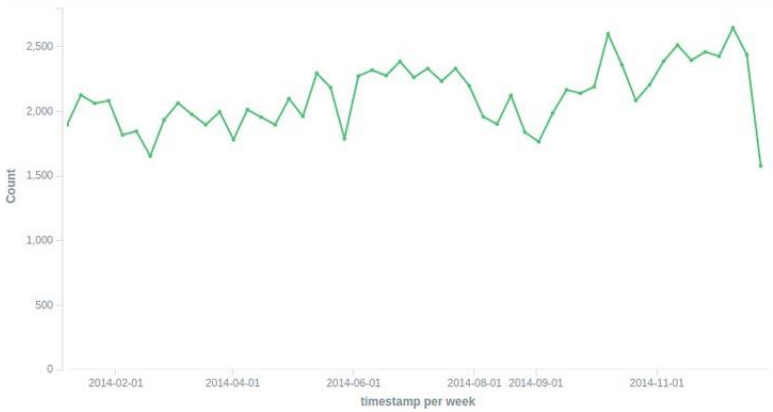

Рисунок 18. График количества аварий за 2014 год

По этому графику можно судить о флуктуации количества аварий по неделям в течении одного года. На основании этого можно сделать вывод, какие периоды в году являются наиболее аварийными.

Рисунок 19. График количества аварий с июня по август 2014 года

По этому графику можно увидеть периодические возрастания и убывания количества аварий с постоянным периодом равным семи дням. Здесь минимумы приходятся на выходные дни, а более высокие значения соответствуют будним дням.

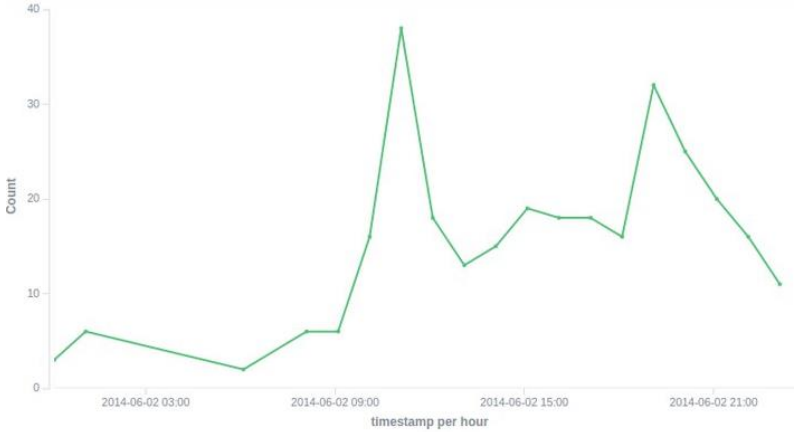

Рисунок 20. График количества аварий за 2 июня 2014 года

На графике аварий за один день можно увидеть два максимума, соответствующих утренним и вечерним часам «пик».

3.1.3 Прочая статистика

Результаты группировки столкновений по различным признакам также удобно визуализировать в Kibana, так как она поддерживает разнообразные диаграммы.

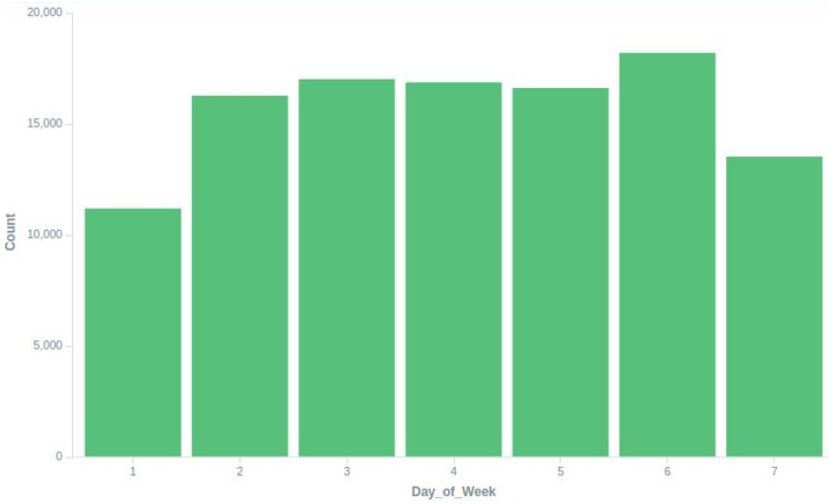

Рисунок 21. Гистограмма распределения количества аварий по дням недели

По этой гистограмме можно увидеть распределение аварий по дням недели. Минимумы здесь приходятся на выходные, а наибольшее число – на пятницу. Цифре 1 здесь соответствует воскресенье, так как неделя в Англии начинается не с понедельника.

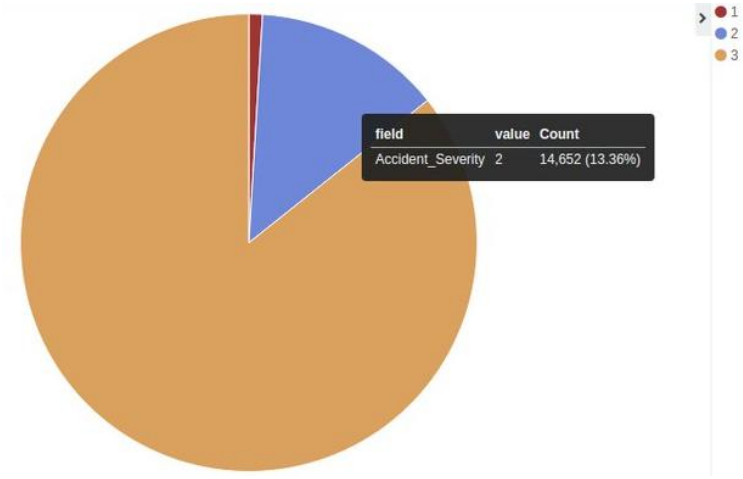

Рисунок 22. Диаграмма распределения аварий по тяжести

На этой круговой диаграмме видно, что большинство произошедших аварий (85 процентов) являются легкими – желтый сектор. Аварии средней тяжести составляют 13 процентов, а тяжелые аварии всего меньше одного процента.

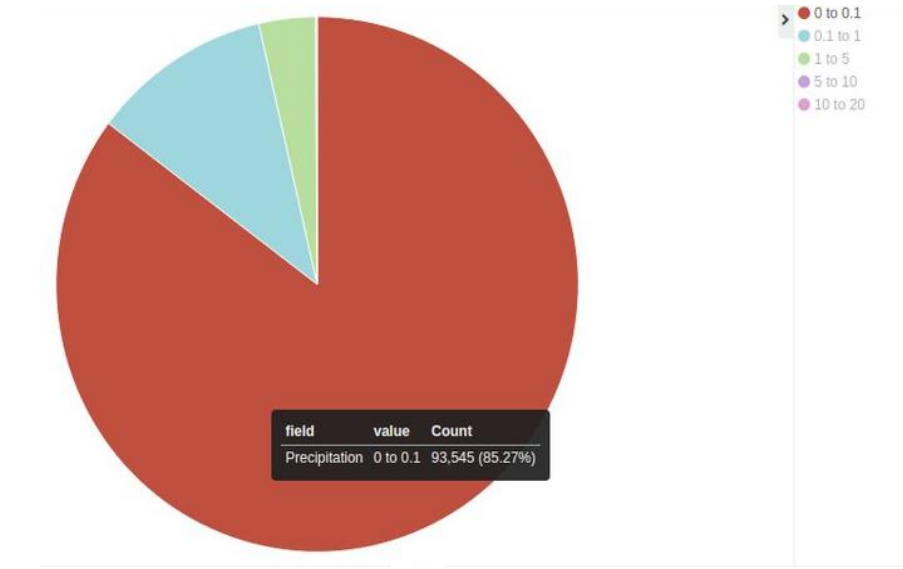

Рисунок 23. Диаграмма распределения количества аварий по количеству осадков

На этой круговой диаграмме видно, что 85 процентов аварий совершались при полном отсутствии осадков.

Kibana позволяет группировать данные по одному признаку. Если требуется группировка по нескольким, это необходимо делать уже вручную, в нашем случае с помощью Pig скриптов. Примером группировки по нескольким признакам может быть выявление групп водителей, попадающих в аварии – по полу, возрасту и месту жительства. В результате этого были определены наиболее аварийноопасные социальные группы и выявлены в следующие особенности:

- Первые четыре места занимают мужчины, проживающие в больших городах и входящими в различные возрастные группы – от 21 до 55 лет
- Значительную часть составляют люди пожилого возраста обоих полов
- Соотнося общее количество женщин и мужчин, попавших в аварии с количеством выданных водительских лицензий, можно увидеть, что женщины попадают в аварии почти в два раза реже чем мужчины – 0,0003 аварии на человека против 0,0007 аварии на человека.

3.2 Прогнозирование

Точность построенной модели определяется самой программой на R и характеризуется коэффициентом детерминации, который составляет 0.7, что означает что модель является приемлемой.

Результатом работы топологии являются сообщения, поступающие в реальном времени и содержащие информацию о спрогнозированном количестве аварий и уровне аварийной опасности. Эти данные записываются в Elasticsearch и визуализируются в Kibana в режиме так называемого «почти реального времени» (near real-time).

Рисунок 24. Результат прогнозирования в Kibana

В таблицу регулярно добавляются новые данные, а старые удаляются с помощью установки и настройки плагина Elasticsearch Curator. Таким образом можно в реальном времени оценивать текущую дорожную ситуацию.

Заключение

В ходе работы были проведен обзор средств работы с большими данными и на их основе разработана система в рамках лямбда архитектуры. Система удовлетворяет всем поставленным требованиям:

- Легко горизонтально масштабируема
- Обладает высокой отказоустойчивостью
- Обладает высоким быстродействием
- Легко расширяема на разные источники данных
- Реализует требуемый функционал

В рамках работы была доказана целесообразность применения технологий работы с большими данными, применительно к области улучшения безопасности дорожного движения. На основании результатов, полученных с помощью этой системы, могут быть предприняты мероприятия по снижению аварийности дорог. Также система эффективна как мониторинговая.

Система обладает большими возможностями и потенциалом к дальнейшему развитию и улучшению.

Таким образом, все поставленные задачи в ходе работы были выполнены, а цель работы можно считать достигнутой.

Список литературы

1. Белокопытов А.В. Смирнов В.Д., Методы корреляционно-регрессионного анализа в эконометрических исследованиях. – Смоленск, 2004. – 150 с.
2. ДеМерс М.Н., Географические информационные системы. Основы – Пер. с англ. Дата+, 1999. – 288 с.
3. Лэм Ч., Hadoop в действии. – М.: ДМК Пресс, 2012. – 424 с.
4. Уайт Т., Hadoop: Подробное руководство. — СПб.: Питер, 2013. — 672 с.
5. Фрэнкс Б., Укрощение больших данных: Как извлекать знания из массивов информации с помощью глубокой аналитики. – М.: Манн, Иванов и Фербер, 2014. – 352 с.
6. Ankit Jain, Anand Nalya. Learning Storm., Packt Publishing, 2014 – 252 с.
7. ESRI White paper ArcGis 3D Analyst. 3D визуализация, топографический анализ, построение поверхностей. – 2002. – 14с.
8. Документация Apache Pig – URL: http://pig.apache.org/docs/r0.15.0/ (дата последнего обращения 12.06.2016)
9. Документация Apache Storm – URL: http://storm.apache.org/releases/0.10.0/index.html (дата последнего обращения 12.06.2016)
10. Документация Elasticsearch – URL https://www.elastic.co/guide/index.html (дата последнего обращения 12.06.2016)