**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**Белорусский государственный Университет**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра многопроцессорных систем и сетей**

ВАТОЛИН

Ростислав Павлович

**КОМПЬЮТЕРНЫЙ МОДУЛЬ КОНТРОЛЯ СИСТЕМЫ ВЕБ-СЛУЖБ НА БАЗЕ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ**

Отчёт о преддипломной практике

студента 5 курса 1 группы

|  |  |  |
| --- | --- | --- |
| “Допустить к защите”  с предварительной оценкой \_\_\_  Руководитель практики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  “\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |  | Научный руководитель:  ассистент кафедры МСС  А.С. Гусейнова |
|  |  |  |

Минск, 2017

**РЕФЕРАТ**

Преддипломная практика, 18 с., 7 рис., 7 источников.

*Ключевые слова*: МЕТЕОРОЛОГИЯ, ВЫСОКОНАГРУЖЕННОЕ ПРИЛОЖЕНИЕ, APACHE STORM, ЗАДАЧА ПРОГНОЗИРОВАНИЯ, МЕТЕОСЛУЖБА, РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ, BIG DATA.

*Объект исследования* – метеорологические веб-сервисы, концепции и методы разработки высоконагруженных приложений, применение фреймворка Apache Storm в создании вычислительного модуля.

*Методы исследования* – обзор и изучение фреймворка Apache Storm, исследование метеорологических сервисов, предоставляющих данные о погоде, получение теоретический и практических знаний в разработке приложений, основанных на распределенных вычислениях.

*Цель работы* – разработка компьютерного модуля на базе фреймворка Apache Storm для контроля веб-метеослужб.

*Результатом* является реализованный модуль контроля веб-служб, предоставляющий достоверного источника и метеорологические данные с высокой точностью, изучение и освоение методов обработки огромного количества данных в реальном времени с помощью фреймворка Apache Storm.

*Областью применения* является метеороло́гия.

[**ВВЕДЕНИЕ 4**](#_Toc476531054)

[1. STORM FRAMEWORK 5](#_Toc476531055)

[**1.1** **Ключевые особенности фреймворка** 5](#_Toc476531056)

[**1.2** **Элементы Storm** 5](#_Toc476531057)

[**1.3** **Сохранность данных** 6](#_Toc476531058)

[**1.4** **Транзакции в Storm** 7](#_Toc476531059)

[2. РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 8](#_Toc476531060)

[**2.1 Получение и сохранение прогноза** 9](#_Toc476531061)

[**2.2 Получение текущего и извлечение сохраненного состояния** 11](#_Toc476531062)

[**2.3 Обработка и отчет** 12](#_Toc476531063)

[ЗАКЛЮЧЕНИЕ 15](#_Toc476531064)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc476531065)

## ВВЕДЕНИЕ

Если следить за прогнозом погоды, то в итоге можно прийти к выводу, что каждый источник будет иметь свою погрешность. В температуре воздуха можно увидеть разницу в пару градусов, во влажности - до 10 процентов, а в атмосферном давлении - до 20 мм рт. ст. Если говорить об атмосферном давлении и влажности, то можно заметить, что расхождения от разных источников значительно отличаются. Неправильные данные об атмосферном давлении, могут причинить дискомфорт людям со слабыми сердечно-сосудистой, дыхательной, центральной нервной системами. Правильные данные о влажности воздуха нужны мамам с младенцами, ведь именно новорожденные очень серьезно воспринимают малейшее изменение окружающей среды.

Постоянный анализ достоверности источников позволит постоянно контролировать качество и точность предоставляемых данных. Имея постоянно работающий компьютерный модуль, который анализирует прогноз и действительные данные в разных точках как городов, так и областей, позволит сказать, какой на данный момент сервис выходит на передний план, среди остальных. Ведь одноразовый анализ и выявление лидера по качеству не будет лучшим решением поставленной задачи.

На сегодняшний день практически невозможно найти подобные механизмы анализа погодных данных. «ОПогоде» - украинский агрегатор погодных данных. Этот сервис решает похожую задачу, которая была описана выше, но имеет ряд недостатков, которые я собираюсь исправить в своем решении. Одним из основных является предоставление данных о погоде в результате усреднения нескольких сервисов, тем самым создавая данные, основываясь как на надежных источниках, так и нет.

В силу высокой загруженности данного модуля отличным решением будет использование фреймворка Apache Storm, написанного на Java и Clojure. Именно с его помощью можно будет осуществить обработку этого колоссально огромного количества данных.

1. **STORM FRAMEWORK**

В 2011 году Twitter открыл, под лицензией Eclipse Public License, проект

распределенных вычислений Storm. Storm был создан в компании BackType и перешел к Twitter после покупки.

Storm это система, ориентированная на распределенную обработку больших потоков данных, аналогичная Apache Hadoop, но в реальном времени.

### **Ключевые особенности фреймворка**

К ключевым особенностям фреймворка Storm относят:

1. Масштабируемость. Задачи обработки распределяются по узлам кластера и потокам на каждом узле;
2. Гарантированная защита от потери данных;
3. Простота развертывания и спровождения;
4. Восстановление после сбоев. Если какой-либо из обработчиков отказывает, задачи переадресуются на другие обработчики;
5. Возможность написания компонентов не только на Java. Простой Multilang protocol с использованием JSON объектов. Есть готовые адаптеры для языков Python, Ruby и Fancy.

### **Элементы Storm**

Основными элементами фреймворка являются:

1. Tuple. Элемент представления данных. По умолчанию может содержать Long, Integer, Short, Byte, String, Double, Float, Boolean и byte[] поля. Пользовательские типы, используемые в Tuple должны быть сериализуемыми;
2. Stream. Последовательность из Tuple. Содержит схему именования полей в Tuple;
3. Spout. Поставщик данных для Stream. Получает данные из внешних источников, формирует из них Tuple и отправляет в Stream. Может отправлять Tuple в несколько разных Stream. Есть готовые для популярных систем обмена сообщениями: RabbitMQ / AMQP, Kestrel, JMS, Kafka;
4. Bolt. Обработчик данных. На вход поступают Tuple. На выход отправляет 0 или более Tuple;
5. Topology. Совокупность элементов с описанием их взаимосвязи. Аналог MapReduce job в Hadoop. В отличии от MapReduce job — не останавливается после исчерпания входного потока данных. Осуществляет транспорт Tuple между элементами Spout и Bolt. Может запускаться локально или загружаться в Storm кластер.
6. **Сохранность данных**

Разные классы задач предъявляют различные требования к надежности. Одно дело пропустить пару записей при подсчете статистики посещений, где счет идет на сотни тысяч и особая точность не нужна. И совсем другое — потерять, например, информацию о платеже клиента.

Рассмотрим о механизмы защиты от потери данных, которые реализованы в Storm.

Если нам не важно были ли ошибки при обработке Tuple, то Spout отправляет Tuple в SpoutOutputCollector посредством вызова метода emit( new Values(...) ).

Eсли необходимо узнать успешно ли обработался Tuple, то вызов будет выглядеть как emit( new Values(...),msgId ).

где - это объект произвольного класса. В этом случае интерфейс ISpout предоставляет методы:

1. - будет вызван, если Tuple обработан успешно;
2. - будет вызван, если Tuple не обработан из-за какой-либо ошибки.

Стоит обратить внимание, что msgId — это id сообщения, с которым был вызван SpoutOutputCollector.emit. Методы nextTuple, ack и fail, вызываются в одном потоке и не требуют дополнительной синхронизации при обращении к полям Spout.

Для того что бы Bolt мог информировать Storm о результатах обработки, он должен реализовывать интерфейс IRichBolt. Проще всего это сделать, унаследовав класс BaseRichBolt. Bolt информирует Storm o результатах своей работы посредством вызова методов класса OutputCollector в методе :

1. – обработка прошла успешно;
2. – обработка завершилась из-за какой-либо ошибки.

В Bolt'ах унаследованных от класса BaseBasicBolt, ack( Tuple tuple ) вызывается после выхода из метода execute автоматически.

При обработке входного Tuple, Bolt может генерировать более одного выходного Tuple. Если Bolt вызвал emit( Tuple sourceTuple,Tuple resultTuple), то образуется направленный ациклический граф (орграф, в котором отсутствуют направленные циклы, то есть пути, начинающиеся и кончающиеся в одной и той же вершине) с вершиной в виде исходного Tuple и потомками в виде порожденных Tuple. Storm отслеживает ошибки процессинга всех узлов графа. В случае возникновения ошибки на любом уровне иерархии, Spout, породивший исходный Tuple, будет уведомлен вызовом fail.

Поскольку Storm является распределенной системой, Tuple могут передаваться с одного узла кластера на другой. В связи с этим Storm обеспечивает отслеживание таймаутов обработки. По умолчанию, весь граф должен быть обработан за 30 секунд, или Storm вызовет метод fail у породившего граф Spout'а. Таймаут можно изменить.

### **Транзакции в Storm**

Представим ситуацию, когда обработка завершилась с ошибкой. Вполне возможно, что потеряна связь с одним из узлов кластера или временно недоступна база данных. В этом случае, нельзя с уверенностью сказать, какие операции выполнились успешно, а какие — нет. Если все операции в цепочке повторно применимы (идемпотентны), например, установка флага, то можно просто перезапустить обработку. Если нет, то на помощь приходят механизмы транзакций Storm.

Когда говорят о характеристиках транзакций, тут же всплывает термин ACID:

1. Atomicity (атомарность). Все изменения, произведенные в системе на протяжении транзакции, либо применяются полностью, либо не применяются совсем;
2. Consistency (cогласованность). Транзакция переводит систему из одного непртиворечивого состояния в другое;
3. Isolation (изолированность). Параллельно выполняемые транзакции не оказывают влияние на результат работы друг друга;
4. Durability (надежность). Зафиксированные транзакцией изменения гарантированно остаются в системе.

В Topology создается объект реализующий интерфейс State, инкапсулирующий работу с БД. Входные данные, поступающие в Spout, разбиваются на Tuple и собираются в пакеты (batch). Batch ассоцируется с уникальным transaction id. Tuple образующие batch могут обрабатываться параллельно.

В конце цепочки обработки, набор Tuple, относящихся к одной транзакции, передается в метод updateState класса, реализующего интерфейс StateUpdater, который и производит модификацию State. В случае успешного завершения, Spout получает уведомление об успехе обработки batch'a. В случае ошибки, Spout должен передать на обработку весь batch повторно.

Таким образом Storm гарантирует, что Batch будет зафиксирован в БД полностью и только один раз.

Storm гарантирует, что Batch'и передаются в StateUpdater строго последовательно, в порядке возрастания transaction id. То есть Batch номер 2 будет зафиксирован только после успешной фиксации Batch'а номер 1.

Транзакционные возможности Storm очень удобно использовать для передачи данных из одной системы в другую, когда требуется нетривиальная обработка. Например, одна система генерирует файлы, Storm их разделяет на записи, обрабатывает в параллельном режиме и складывает в БД. В случае ошибки обработки есть гарантия, что файл не будет удален и не будет обработан дважды.

1. **РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**

В качестве проекта было решено реализовать модуль контроля погодный веб-служб, основанных на протоколе HTTP, использующие REST архитектуру передачи данных. Были разработаны компоненты фреймворка Apache Storm согласно рисунку 2.1. Вместе они работают для сравнения текущих погодных данных с данными на час вперед.

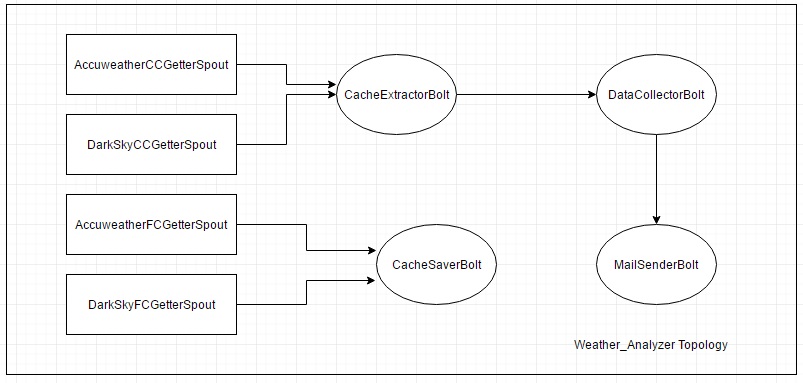


Рисунок 2.1 – Топология реализованного модуля

### **2.1 Получение и сохранение прогноза**

Для получения прогноза на час вперед в данном модуле отвечают DarkSkyFCGetterSpout и AccuweatherFCGetterSpout. Каждый из них обращается к соответствующей веб-службе за погодными данными один раз в час. Получение метеоданных происходит независимо в разных потоках данного модуля.

Веб-службы accuweather.com и darksky.net предоставляет REST API с возможностью делать 1000 запросов в день бесплатно. На рисунке 2.2 и на рисунке 2.3 представлены результаты запросов о метеоданных на час вперед в городе Минске.

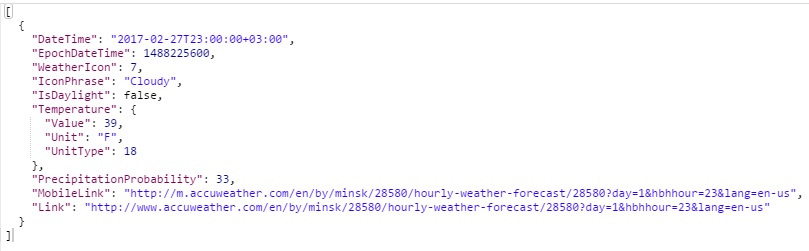


Рисунок 2.2 – прогноз на час от accuweather.com

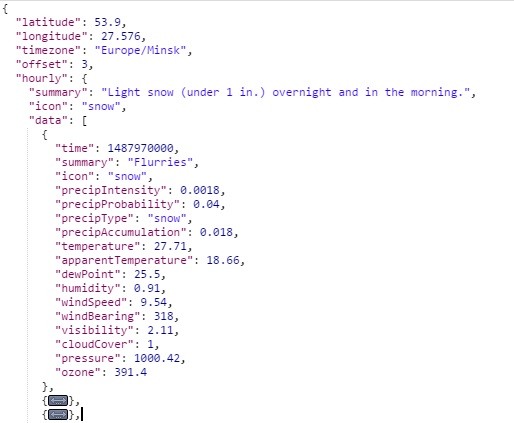


Рисунок 2.3 – прогноз от darksky.net

Как можно заметить, каждая веб-служба имеет свою модель данных, поэтому было принято решение о создании собственной, общей для всех структуры, с которой будет проще и удобнее взаимодействовать в дальнейшем. Общая модель представлена на рисунке 2.4.

Рисунок 2.4 – модель объекта общего типа

После получения данных о прогнозе и преобразовании их к объекту модели общего вида, идет передача их в CacheSaverBolt. На этом этапе поступившему объекту создается ключ, затем он целиком помещается в кэш.

**2.2 Получение текущего и извлечение сохраненного состояния**

Для получения текущего состояния погоды используются AccuweatherCCGetterSpout и DarkSkyCCGetterSpout. Обращение к веб-службам происходит один раз в час. На рисунке 2.5 и рисунке 2.6 представлены результаты запросов о метеоданных на текущее состояние в городе Минск.



Рисунок 2.5 – текущее состояние от accuweather.com

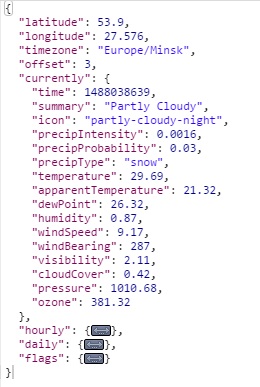


Рисунок 2.6 – текущее состояние от darksky.net

После успешного преобразования в объект общей модели, идет его передача в CacheExtractorBolt, где происходит поиск и извлечения из кэша прогноза, который был помещен туда ровно час назад. Если объект будет найден, то произойдет передача данных о прогнозе и о текущем состоянии в DataCollectorBolt для дальнейшего анализа и генерации отчета.

**2.3 Обработка и отчет**

В случае, когда в DataCollectorBolt пришли два соответствующих объекта модели WeatherConditionTO, создается оценка погрешности для данного города от соответствующей веб-службы. Оценка сохраняется в кэш в виде объекта LocationAccuracy, модель которого изображена на рисунке 2.7. Затем формируется объект для будущего включения в отчет, его модель изображена на рисунке 2.8.

Рисунок 2.7 – модель объекта оценки точности прогноза за час

Рисунок 2.8 – модель объекта для генерации отчета

Для создания отчетности и отправки ее на почту, указанную в настройках отвечает MailSenderBolt. Один раз в час он достает объекты типа HourAccuracy и создает таблицу в формате csv, которую в дальнейшем можно будет открыть с помощью Microsoft Excel. Данный файл будет отправлен на почту ответственному лицу. Пример содержимого файла csv изображено на рисунке 2.9.

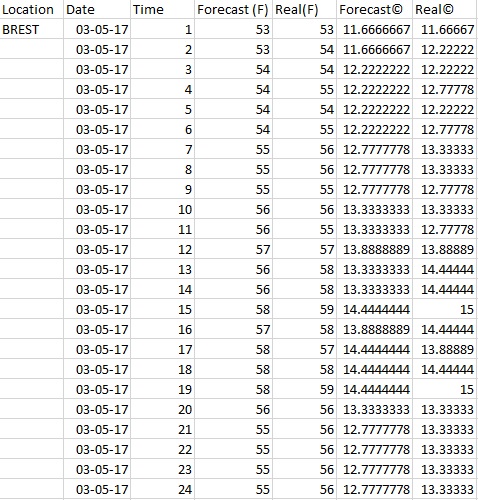


Рисунок 2.9.1 – пример отчета по погоде, отправленный на почту

Нетрудно вывести диаграмму, согласно рисунку 2.9.1– рисунок 2.9.2

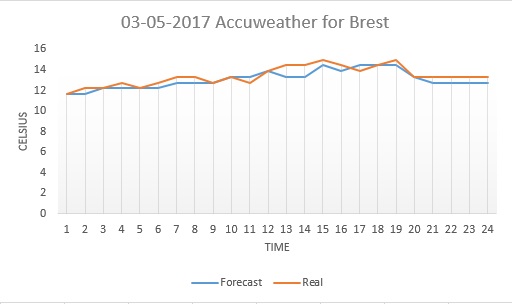


Рисунок 2.9.2 – диаграмма по рисунку 2.9.1

**ЗАКЛЮЧЕНИЕ**

В данной работе:

* изучен фреймворк Apache Storm и взаимодействие его компонентов;
* изучена работа с отправкой писем на электронные почтовые ящики;
* рассмотрен спектр погодных веб-служб, предоставляющих бесплатный доступ к метеоданным;
* разработан высоконагруженный модуль получения, обработки, и предоставления отчета о данных, полученных от метеослужб.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Эккель Б. Философия Java. Библиотека программиста. – СПб.: Питер, 2001. – 880 с.: ил;
2. Буч Г. Объектно-ориентированный анализ и проектирование. 2-е изд. Пер. с англ. – М.: «Издательство Бином», 1998 – 560с.: ил;
3. Хабибуллин И. Ш. Создание распределённых приложений на Java 2. – СПб.: БХВ-Петербург, 2001 г. – 688с.: ил;
4. Java. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с;
5. Шаблоны проектирования в Java / М. Гранд; Пер. с англ. С. Беликовой. – М.: Новое знание, 2004. – 559 с.: ил;
6. Материалы ресурса Habrahabr [Электрон. ресурс] Habrahabr Project – Режим доступа: https://habrahabr.ru/ – Дата доступа: 16.02.2017;
7. Материалы свободной энциклопедии Wikipedia [Электрон. ресурс] / Wikipedia Project – Режим доступа: http://wikipedia.org – Дата доступа: 02.02.2017.