**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**Белорусский государственный Университет**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра многопроцессорных систем и сетей**

ВАТОЛИН

Ростислав Павлович

**РЕАЛИЗАЦИЯ МОДУЛЯ КОНТРОЛЯ СИСТЕМЫ ВЕБ-СЛУЖБ НА БАЗЕ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ**

Дипломная работа

|  |  |
| --- | --- |
|  | Научные руководители:  Ассистент кафедры МСС  А.С. Гусейнова  Зав. Кафедрой МСС  доцент, кандидат физико-математических наук  С.В. Марков |

Допущена к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

Зав. кафедрой МСС

доцент, кандидат физико-математических наук С. В. Марков

Минск, 2017

**РЕФЕРАТ**

Дипломная работа, 40 с., 24 рис., 1 табл., 9 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 исследования***– создание программного модуля с применением фреймворка Apache Storm для контроля веб-метеослужб для выявления лидера в точности прогноза.

***В результате исследования*** реализован модуль контроля веб-служб, предоставляющий достоверный источник метеорологических данных с высокой точностью, изучен и освоен ряд методов обработки огромного количества данных в реальном времени с помощью фреймворка Apache Storm.

***Область применения*** – метеороло́гия.

**РЭФЕРАТ**

Дыпломная праца, 40 с., 24 мал., 1 табліца, 9 крыніц.

***Ключавыя словы***: МЕТЭАРАЛОГIЯ, ВЫСАКАНАГРУЖЕННАЕ ПРЫКЛАДАННЕ, APACHE STORM, ЗАДАЧА ПРАГНАЗАВАННЯ, МЕТЭОСЛУЖБА, РАЗМЕРКАВАНЫЯ ВЫЛИЧЭННI, BIG DATA.

***Аб'ект даследавання*** – метэаралагiчныя вэб-сэрвiсы, канцэпцыi i метады распрацоўкi высаканагружаных прыкладанняў, прымяненне фреймворка Apache Storm у стварэннi вылiчальнага модуля.

***Мэта даследавання*** – стварэнне праграмнага модуля з ужываннем фрэймворка Apache Storm для кантролю вэб-метэослужб для выяўлення лiдэра ў дакладнасцi прагнозу.

***У выніку даследавання*** рэалiзаваны модуль кантролю вэб-службаў, якi прадстаўляе пэуную крынiцу метэаралагiчных дадзеных з высокай дакладнасцю, вывучаны i засвоены шэраг метадаў апрацоўкi велiзарнай колькасцi дадзеных у рэальным часе з дапамогай фреймворка Apache Storm.

***Метады даследавання*** – абзор i вывуэнне фрэймворка Apache Storm, даследаванне метеаралагiчных вэб-служб, якiя прадастаўляюць дадзеныя аб надвор’i, атрыманне тэарэтычных i практычных ведаў у распрацоўцы прыкладанняў, заснаваных на размеркаваных вылiчэннях.

***Вобласць прымянення*** – метэаралогiя.

**ABSTRACT**

Thesis, 40 p., 24 fig., 1 table, 9 sources.

***Keywords:*** METEOROLOGY, HIGHLY LOADED APPLICATION, APACHE STORM, PREDICTION PROBLEM, METEOROLOGICAL SERVICE, DISTRIBUTED CALCULATIONS, BIG DATA.

***The object of study*** – meteorological web services, concepts and methods for developing highly loaded applications, the use of the Apache Storm framework in the creation of a computational module.

***The purpose of work*** – creation of a software module using the Apache Storm framework to monitor web meteorological services to identify the leader in the accuracy of the forecast.

***Research methods*** – the study of the Apache Storm framework, the study of meteorological services providing weather data, obtainment of theoretical and practical knowledge in the development of applications based on distributed computing.

***As a result,*** a module of web service control was created, that provide a reliable source of meteorological data with high accuracy, studied and mastered a number of methods for processing a huge amount of data in real time using the Apache Storm framework.

***Scope*** – meteorology.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 7](#_Toc483170505)

[JAVA 8](#_Toc483170506)

[**1.1 Введение** 8](#_Toc483170507)

[**1.2 Уникальность Java** 8](#_Toc483170508)

[**1.3 Платформа** 11](#_Toc483170509)

[STORM FRAMEWORK 13](#_Toc483170510)

[**2.1 Ключевые особенности фреймворка** 13](#_Toc483170511)

[**2.2 Элементы Storm** 13](#_Toc483170512)

[**2.3 Сохранность данных** 14](#_Toc483170513)

[**2.4 Транзакции в Storm** 15](#_Toc483170514)

[EHCACHE 17](#_Toc483170515)

[**3.1 Введение в Ehcache** 17](#_Toc483170516)

[**3.2 Терминология** 17](#_Toc483170517)

[**3.3 Уровни хранения данных** 19](#_Toc483170518)

[**3.4 Улучшения производительности** 21](#_Toc483170519)

[РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ 23](#_Toc483170520)

[**4.1 Анализ доступных погодных веб-служб** 23](#_Toc483170521)

[**4.2 Анализ спектра предоставляемых данных** 25](#_Toc483170522)

[**4.3 Локации и частота выполняемых измерений** 28](#_Toc483170523)

[**4.4 Оценивание точности прогнозов** 29](#_Toc483170524)

[**4.5 Предоставляемые отчеты** 30](#_Toc483170525)

[РЕЗУЛЬТАТЫ И АНАЛИЗ 33](#_Toc483170526)

[**5.1 Рассмотрение полученных писем-отчетов** 33](#_Toc483170527)

[**5.2 Рассмотрение веб-служб по локациям** 38](#_Toc483170528)

[ЗАКЛЮЧЕНИЕ 39](#_Toc483170529)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc483170530)

# **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 планируется исправить в данной работе. 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.

**JAVA**

**1.1 Введение**

Java – это независимый от платформы высокоуровневый язык программирования. Он был создан для работы в распределенной среде в Интернете. Java имеет функции GUI, которые лучше

воспринимаются, чем в языке C++, кроме того, он проще в использовании, чем C ++, и работает над концепцией модели объектно-ориентированного программирования. Также широко используется в онлайн играх,

помогает передовать видео, аудио, общаться с людьми по всему миру, существует масса банковских приложений, помогает просматривать 3D-изображения и корзины покупок. Java находит широкое применение

в приложениях интранет и других решениях для электронного бизнеса, в которых присутствуют массовые корпоративные вычисления. Рассматривается как наиболее хорошо описанный и планируемый язык

для разработки приложений для Интернета.

Java – это хорошо известная технология, которая позволяет вам иметь программное обеспечение, предназначенное и написанное только один раз для «виртуальной машины» для работы на разных компьютерах,

поддерживает различные операционные системы, такие как компьютеры под управлением Windows, Macintosh и Linux. В веб-аспекте Java популярна на веб-серверах, используемых многими крупнейшими

интерактивными веб-сайтами. Она используется для создания автономных приложений, которые могут выполняться на одном компьютере или в распределенной сети. Java также может быть использована для

создания небольшой прикладной программы на основе апплета, которая далее используется для веб-страницы. Аплеты упрощают взаимодействие с веб-страницей.

**1.2 Уникальность Java**

Являясь языком высокого уровня, он обладает такими качествами, как:

1. объектно-ориентированность;
2. простота;
3. архитектурная нейтральность;
4. портативность;
5. распределенность;
6. высокопроизводителен;
7. обеспечивает многопоточность;
8. надежность;
9. динамичность;
10. безопасность.

Java имеет преимущество перед другими языками и средами, которые делают его пригодным для любой задачи программирования. Выделяют следующие преимущества:

1. Java сам по себе прост, прост в дизайне, удобен в написании и поэтому легко компилируется, отлаживается и учится, чем любые другие языки программирования;
2. Java объектно-ориентированна, используется для построения модульных программ и кода многократного использования;
3. Java является платформо-независимым и гибким по своей природе. Самой важной особенностью Java является возможность легко запускать программу как в одной компьютерной системе, так и в другой;
4. Java работает в распределенной среде. Он предназначен для работы в распределенных вычислениях. Любые сетевые программы на Java аналогичны программам на отправку и прием данных в и из файла;
5. Java является безопасным. Язык Java, компилятор, интерпретатор и среда выполнения являются защищаемыми;
6. Java является надежной. В Java есть способна проверять код на наличие возможных ошибок на этапе компиляции;
7. Java поддерживает многопоточность. Многопоточность - это путь выполнения программы, в которой выполняются нескольких задач одновременно. В Java реализована концепция многопоточной программы. На других языках для работы в многопоточном режиме необходимо вызвать специальные для операционной системы процедуры.

Java - это язык программирования высокого уровня и одновременно мощная программная платформа. При полном использовании платформы Java вы получаете следующие возможности:

* 1. Средства JDK. Инструменты JDK обеспечивают компиляцию, интерпретацию, запуск, мониторинг, отладку и документирование ваших приложений. Основными используемыми инструментами являются компилятор Javac, java launcher и инструмент документации Javadoc;
  2. Интерфейс прикладного программирования (API): API обеспечивает основные функциональные возможности языка программирования Java. Он предоставляет широкий набор полезных классов, которые далее используются в ваших собственных приложениях. Он предоставляет базовые объекты и интерфейс для сетей и безопасности, для генерации XML и доступа к базе данных и многое другое;
  3. Технологии развертывания. Программное обеспечение JDK предоставляет два типа технологий развертывания, такие как программное обеспечение Java Web Start и программное обеспечение Java Plug-In для развертывания ваших приложений для конечных пользователей;
  4. Инструментарий графического пользовательского интерфейса: набор инструментов Swing и Java 2D предоставляет нам функцию графических пользовательских интерфейсов (GUI);
  5. Интегрированные библиотеки: реализована интегриация с различными библиотеками, такими как Java IDL API, API JDBC, API для Java Naming и Directory Interface TM («JNDI»), Java RMI и Java Remote Method Invocation по технологии Inter-ORB протокола Интернета (Java RMI-IIOP технология) позволяют базе данных получать доступ и изменять удаленные объекты.

Java платформа меняет нашу жизнь. Легко начать пользоваться. Поскольку язык программирования Java полностью основан на объектно-ориентированном языке, его очень легко и просто освоить, особенно для программистов, уже знакомых с C или C ++. Легко начать писать код: по сравнению с метриками программы (подсчет классов, количество методов и т. д.) программа, написанная на языке программирования Java, может быть в четыре раза меньше по сравнению с той же программой, написанной на C ++. Язык программирования Java поощряет хорошие методы кодирования и управляет автоматическим сбором мусора, который помогает избежать утечек памяти. Основанная на концепции ориентации объектов, ее компонентная архитектура Java Beans и широкая область применения, легко расширяема, гибкая и API могут повторно использовать существующий, протестированный код и создавать меньше ошибок. Java проще, чем C ++, как таковой, управляет вашим временем разработки в два раза быстрее, когда пишете на нем. Для программ также потребуется меньшее количество строк кода. Независима от платформы: программа сохраняет портативность и независимость от платформы, избегая использования библиотек, написанных на других языках. Любой исходный код программы, написанный на языке программирования Java, который компилируется в машинно-независимые байтовые коды и выполняется последовательно на любой платформе Java. Распространение программного обеспечения упрощает работу: используя программное обеспечение Java Web Start, пользователи смогут запускать собственные приложения одним щелчком мыши.

**1.3 Платформа**

Платформа представляет собой кросс-сочетание аппаратного и программного окружения, в котором выполняется программа. Мы уже знакомы с самой популярными платформами, такими как Microsoft Windows, Linux, Solaris OS и Macintosh OS. Платформа Java отличается от других платформ, которая является единственной программной платформой, которая работает на других аппаратных платформах.

Платформа Java состоит из двух компонент (рисунок 1.1): виртуальная машина (JVM) и интерфейс прикладного программирования (API). Виртуальная машина Java является корнем платформы Java и интегрирована в различные аппаратные платформы.

API представляет собой обширную коллекцию различных программных компонент, которые предоставляют множество полезных функций для приложения. Он сгруппирован в логическую коллекцию связанных классов и интерфейсов. Эти логические коллекции называются пакетами.

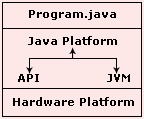


Рисунок 1.1 – java платформа

API и виртуальная машина Java изолируют программу от аппаратного обеспечения. Java работает на платформо-независимой среде, сама платформа немного медленнее, чем собственный код. Однако новые изменения в компиляторах и виртуальных машинах приближают производительность к естественному коду, не представляя угрозы безопасности для переносимости. Весь исходный код на Java находится в текстовых файлах с расширением .java. Исходные файлы компилируются в файлы .class компилятором. Файл .class содержит байт-код, который в дальнейшем будет выполнен с помощью виртуальной машины Java (JVM). Инструмент запуска Java запускает приложение с экземпляром виртуальной машины Java, как видно на рисунке 1.2.

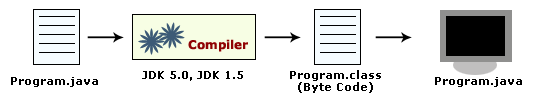


Рисунок 1.2 – создание байт-кода

JVM запускает байт код в любой системе, то соответственно она и работает в любой операционной системе одинаково. Для этого создаются определенные версии JVM, рассчитанные для работы в необходимой среде. К примеру, виртуальная машина Java Hotspot, рассчитанная для систем Microsoft Windows, Linux, Solaris, повышает производительность приложения во время выполнения, благодаря поиску “горячих мест”, которые очень часто выполняются. В ее обязанности входят различные задачи, такие как повышение эффективности программы и перекомпиляция (в машинный код) исходного кода, который часто используется при выполнении программы (рисунок 1.3).

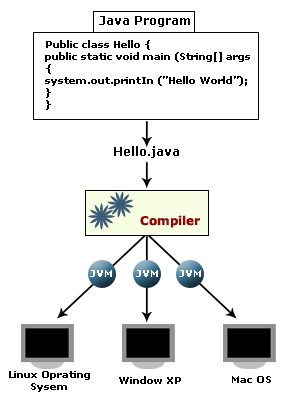


Рисунок 1.3 – исполнение java кода на различных системах

На изображении выше довольно доходчиво показано основное преимущество программ, написанных на java. Именно JVM, является тем слоем в данной схеме, который гарантирует работу приложений на различных операционных системах.

**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.

### **2.1 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.

### **2.2 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.

**2.3 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.

### **2.4 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.

**EHCACHE**

**3.1 Введение в Ehcache**

Ehcache - основанный на стандартах кеш с открытым исходным кодом. Создан для повышения производительности, разгрузки баз данных и упрощения масштабирования программного продукта. Являясь надежным, проверенным и полнофункциональным решением, он наиболее широко используем на сегодняшний день в приложениях, написанных на Java. Можно использовать Ehcache как кеш общего назначения или как кэш второго уровня для Hibernate. Также можно дополнительно интегрировать его с продуктами сторонних производителей, такими как ColdFusion, Google App Engine и Spring.

Ehcache предоставляет кэш в процессе, который можно реплицировать на нескольких узлах. Он также лежит в основе BigMemory Go и BigMemory Max, коммерческого кэширования Terracotta и хранения данных в памяти. Массив Terracotta Server Array, поставляемый с BigMemory Max, позволяет использовать смешанные конфигурации в процессе или вне процесса с кэшами размером в терабайт.

**3.2 Терминология**

**Cache** - википедия определяет этот термин как “a store of things that will be required in the future, and can be retrieved rapidly.”, что переводится, как хранилище вещей, которые потребуются в будущем, и которые могут быть быстро извлечены. Сам по себе кэш представляет собой набор временных данных, которые либо дублируют данные, расположенные в другом месте, либо являются результатом вычисления. Данные, которые уже находятся в кэше, могут быть повторно доступны с минимальными затратами с точки зрения времени и ресурсов.

**Cache hit** - когда элемент данных запрашивается из кэша и он существует для данного ключа, то это случай носит название “кэш попадание” (cache hit).

**Cache miss** - когда элемент данных запрашивается из кэша и он не существует для данного ключа, то данный случай носит название “кэш промах“ (cache miss).

**System of Record (SOR)** - источник кэшированных данных. Кэш действует как локальное хранилище копий данных, полученных или сохраненных в системе записи (SOR). SOR часто является традиционной базой данных, хотя это может быть специализированная файловая система или какое-либо другое надежное долговременное хранилище. Работая с Ehcache, предполагается, что в качестве SOR используется база данных.

**Standalone cache topology** - автономная кэш топология, при которой данные хранятся в узле приложения. Любые другие узлы приложения независимы, между ними нет связи. Если используется автономная топология, когда несколько узлов используют одно и то же приложение, то между ними существует слабая согласованность. Они содержат согласованные значения для неизменяемых данных или обновляют элемент данных после того, как время жизни элемента завершилось.

**Distributed cache topology** - распределенная кэш топология, данные хранятся на удаленном сервере (или кластере серверов) с поднабором недавно используемых данных, хранящихся в каждом узле приложения. Эта топология предлагает богатый набор параметров для согласованности. Распределенная топология является рекомендуемым подходом в кластерной или масштабируемой прикладной среде. Она обеспечивает высочайший уровень производительности, доступности и масштабируемости. Распределенная топология доступна только с BigMemory Max.

**Replicated cache topology** - реплицированная кэш топология, кэшированный набор данных хранится в каждом узле приложения, и данные копируются или недействительны по узлам без блокировки. Репликация может быть асинхронной или синхронной, где поток записи блокируется во время распространения. Единственный режим согласованности, поддерживаемый в этой топологии, - «Слабая согласованность». Многие производственные приложения развертываются в кластерах из нескольких экземпляров для обеспечения доступности и масштабируемости. Однако без распределенного или реплицированного кэша кластеры приложений демонстрируют ряд нежелательных действий, таких как cache drift и database bottlenecks.

**Cache Drift** - в этом случае, каждый экземпляр приложения хранит свой собственный кеш. Обновления, сделанные в одном кэше, не будут отображаться в других экземплярах. Это происходит с данными веб-сессии. Распределенная или реплицированная топологии кэша обеспечивает синхронизацию всех экземпляров кэша друг с другом.

**Database Bottlenecks** - узкие места в базе данных. В одиночном экземпляре приложения, кэш эффективно защищает базу данных от избыточных запросов. Однако в распределенной среде приложений каждый экземпляр имеет большую нагрузку и обновляет свой собственный кэш. Издержки при загрузке и обновлении нескольких кэшей приводят к узким местам базы данных по мере добавления большего количества экземпляров приложений. Распределенный или реплицированный кэш устраняет накладные расходы для каждого экземпляра загрузки и обновления нескольких кэшей из базы данных.

**Locality of reference** - в информатике - местность ссылки, также известная как принцип локальности, является термином для явления, в котором часто используются одни и те же значения или связанные с ним места хранения, в зависимости от шаблона доступа к памяти. Существует два основных типа эталонного местоположения - временная и пространственная локальность. Временная локальность относится к повторному использованию определенных данных и / или ресурсов в течение относительно небольшой продолжительности времени. Пространственная локальность относится к использованию элементов данных в относительно близких местах хранения. Последовательная локальность - частный случай пространственной локальности - возникает, когда элементы данных упорядочиваются и получают линейный доступ, например, обход элементов в одномерном массиве.

**3.3 Уровни хранения данных**

Вы можете разделить кэш или данные в памяти в следующих областях хранения, называемых уровнями:

1) MemoryStore - память в куче, используемая для хранения элементов кэша. Этот уровень может быть подвержен сборке мусора со стороны Java. Хранилище памяти всегда включено и существует в куче памяти. Оно принимает все данные, будь-то сериализуемые они или нет. Также стоит упомянуть, что это самый быстрый вариант хранения и является потокобезопасным для использования несколькими параллельными потоками. Если вы используете OffHeapStore (доступен только в продуктах BigMemory), MemoryStore хранит копию самого горячего подмножества данных из OffHeapStore.

Все кеши определяют максимальный размер в памяти, с точки зрения количества элементов, во время настройки. Когда элемент добавляется в кеш и выходит за пределы его максимального объема памяти, существующий элемент либо удаляется, если переполнение не активировано, либо оценивается для буферизации на другом уровне, если переполнение включено. Если переполнение включено, выполняется проверка на истечение срока действия хранимых данных. Если срок вышел, то он удаляется, иначе он остается сохраненным, но в отдельном месте;

2) OffHeapStore - предоставляет возможность переполнения памяти MemoryStore. Ограничено по размеру только доступной оперативной памятью. Не подлежит сборке мусора Java (GC). Доступно только для продуктов Terracotta BigMemory. OffHeapStore расширяет кеш к памяти за пределами кучи Java. Этот хранилище, не подлежит воздействию со стороны сборщика мусора Java (GC), ограничено только объемом доступной памяти. Используя OffHeapStore, можно создавать очень большие локальные кеши. OffHeapStore доступен только для продуктов Terracotta BigMemory.

Поскольку данные вне кучи хранятся в байтах, только данные, которые являются Serializable, подходят для OffHeapStore. Любые несериализуемые данные, переполняемые в OffHeapMemoryStore, просто удаляются, и выдается сообщение логгера уровня WARNING. Поскольку сериализация и десериализация происходят при переносе и получении из хранилища не-кучи, теоретически она медленнее, чем MemoryStore. Однако это различие смягчается, когда учитывается сборка мусора, связанная с большими кучами. Для достижения наилучшей производительности рекомендуется выделять для кэша как можно больше памяти кучи, не запуская GC-паузы. Затем используйте OffHeapStore для хранения данных, которые не умещаются в куче (не вызывая GC-паузы);

3) DiskStore - создает резервные копии элементов кэша в памяти и обеспечивает переполнение для других уровней. DiskStore обеспечивает потокобезопасное средство буферизации дисков, которое может быть использовано для дополнительного хранения или сохранения данных при перезагрузке системы. Стоит заметить, что уровень DiskStore доступен только для локальных (автономных) экземпляров кэша. Когда вы используете распределенный кеш (доступный только в BigMemory Max), Terracotta Server Array используется вместо уровня диска. Только данные Serializable могут быть помещены в DiskStore. Записывает и использует диск ObjectInputStream и механизм сериализации Java. Любые несериализуемые данные, переполняющие хранилище на диске, удаляются и генерируется исключение NotSerializableException. Стоить иметь в виду, что на скорость сериализации влияет размер сериализуемых объектов и их тип. Например, было показано, что время сериализации для Java-объекта, состоящего из большого массива Map String, составляет около 126 мс, где сериализованный размер получается 349,225 байт. Также время сериализации для массива байтов составило 7 мс, где сериализованный размер около 310,232 байта. Байт-массивы в 20 раз быстрее сериализовать, что делает их лучшим выбором для повышения производительности дискового хранилища. Конфигурирование хранилища дисков не является обязательным. Если все кэши используют только память и хранилища не-кучи, нет необходимости настраивать хранилище дисков. Это упрощает настройку и использует меньше потоков.

**3.4 Улучшения производительности**

Хоть, Ehcache относится к объектам Java, кэширование используется во всех вычислениях, от кэшей ЦП до системы доменных имен в Интернете (DNS). Это происходит из-за того, что многие компьютерные системы показывают «местность ссылки». Данные, которые находятся рядом с другими данными или недавно использовались, с большей вероятностью будут использоваться снова.

Крис Андерсон из Wired Magazine придумал термин «длинный хвост» (рисунок 3.1) для обозначения случаев в системах электронной коммерции, где небольшое количество товаров может составлять основную часть продаж (или небольшое количество блогов может получить наибольшее количество обращений).



Рисунок 3.1 – длинный хвост

И как следствие есть длинный «хвост» менее популярных предметов. Длинный хвост является примером закона силы распределения вероятности, такого как распределение Парето или правило 80:20. Если 20 процентов объектов используется в 80 процентах случаев и можно найти способ снизить затраты на получение этих 20 процентов, производительность системы улучшится.

Стоит задать вопрос о том, на сколько приложение получает преимущество от кэширования. Преимущество особенно заметно, когда есть работа с вводом-выводом и есть нужда в скорости, с которой данные могут быть получены. Если же имеем дело с CPU, тогда время зависит от скорости центрального процессора и основной памяти. Кэширование обычно повышает производительность и снижает нагрузку на веб-сервер.

Приложения с плотной привязкой к CPU, часто ускоряются за счет:

1) улучшения производительности алгоритма;

2) распараллеливания вычислений между несколькими процессорами (SMP) или несколькими машинами (кластерами);

3) увеличения скорости процессора.

Кэш может временно хранить вычисления для повторного использования, включая, но не ограничиваясь большими веб-страницами с высокой стоимостью рендеринга или состояниями аутентификации, когда аутентификация требует криптографических преобразований.

Многие приложения связаны с операциями ввода-вывода, либо дисковыми или сетевыми операциями. В случае баз данных они могут быть ограничены обоими.

Для жестких дисков нет закона Мура. 10 000 оборотов в минуту были быстрыми 10 лет назад и по-прежнему быстры. Жесткие диски ускоряются, используя собственное кэширование блоков в память.

Сетевые операции могут быть замедленны благодаря следующим факторам:

1) время для установки и разрыва соединения;

2) задержка, или минимальное время прохода туда и обратно;

3) пределы пропускной способности;

4) временные затраты на преобразования.

Кэширование является распространенным выходом для повышения производительности приложений, особенно тех, которые постоянно выполняют операции по вводу-выводу. Ehcache очень удобно использовать для кэширования объекта доступа к данным для Hibernate или для кэширования веб-страниц, для страниц, созданных из баз данных.

Следствием повышения производительности является повышенная масштабируемость. Если предположим, что у нас есть база данных, которая может выполнять до 100 дорогих запросов в секунду. Помимо данного порога, база данных резервирует запросы, и если возникают дополнительные соединения, производительность работы с базой данных медленно падает. В этом случае кэширование, вероятно, уменьшит рабочую нагрузку. Если кэширование может привести к тому, что 90% из этих 100 будут хитами кэша и не повлияют на базу данных, база данных может масштабироваться в 10 раз.

**РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ**

### **4.1 Анализ доступных погодных веб-служб**

Разрабатывая высоконагруженный модуль очень важно выбрать надежных поставщиков данных, которые бесперебойно будут возвращать ожидаемый результат. В то же время, не имея достаточного бюджета для покупки погодного пакета услуг у лидирующих веб сервисов, в рассмотрение были включены хорошо зарекомендованные себя в сферах стартапов погодные службы, о которых пойдет речь дальше.

Первым был рассмотрен погодный портал AccuWeather. Он является одним из ведущих поставщиков цифровой информации о погоде. Согласно его веб-сайту, AccuWeather предоставляет прогнозы погоды для почти 3 миллионов местоположений во всем мире, и более миллиарда людей во всем мире полагаются на AccuWeather каждый день.

Компания постоянно работает над расширением своей аудитории, а также добавлением новых продуктов и услуг, связанных с погодой. В конце прошлого года компания приобрела Sky Motion, разработчика экстремальных краткосрочных и сильно локализованных прогнозов погоды. Приобретение Sky Motion позволило AccuWeather запустить функцию MinuteCast, которая была представлена ​​в начале этого года. В феврале AccuWeather объявила о партнерстве с Samsung, чтобы обеспечить предустановленное приложение погоды на смартфоне Samsung Galaxy S5. Компания также планирует запустить канал AccuWeather Channel, 24/7.

Разработчики могут использовать AccuWeather APITrack этого API для добавления суровых погодных предупреждений, ежедневных и ежечасных прогнозов, текущих условий и других связанных с погодой функций в свои приложения. Для некоторых возможностей версия freemium не доступна; AccuWeather предоставляет практически все свои сервисы только за купленные планы. Разработчики, заинтересованные в использовании API AccuWeather, могут связаться с отделом продаж компании по вопросам ценообразования.

Рассматривая погодные веб-службы, нельзя обойти и Австралийское бюро метеорологии. Оно отвечает за оказание австралийцам помощи в борьбе с засухой, наводнениями, пожарами, штормами и другими суровыми погодными условиями. Агентство также предоставляет разнообразные данные о погоде, климате и воде, включая наблюдения в режиме реального времени, прогнозы компьютерной модели и сельскохозяйственные бюллетени.

В марте бюро запустило веб-сайт Наблюдения за погодой (WOW) и проект Met Office. Проект WOW является еще одним примером краудсорсинга погоды. Австралийские энтузиасты погоды могут обмениваться метеорологическими наблюдениями, фотографиями и другой информацией, используя веб-сайт WOW (сайт WOW можно использовать бесплатно).

Еще одним примером качественного предоставления данных можно смело выделить Forecast.io. Эта служба погоды, созданная с нуля и созданная для приложения Dark Sky Weather для iPhone и iPad. В марте 2013 года Forecast.io был запущен какотдельная служба погоды Forecast APITrack, этот API был предоставлен разработчикам, чтобы они могли включать метеорологические данные и функциональные возможности Forecast.io в сторонние приложения. Согласно веб-сайту, тысячи разработчиков подписались на использование API Forecast. Приложения и веб-сайты, использующие данные о погоде Forecast.io, включают DuckDuckGo, приложение Saga, приложение Weather Line и приложение Dark Sky.

API Forecast можно использовать для получения метеорологических данных, включая текущие условия, поминутные прогнозы, ежечасные прогнозы и ежедневные прогнозы. API может использоваться как для некоммерческих, так и для коммерческих приложений, а 1000 вызовов API в день максимум бесплатны, что делает ее очень хорошим вариантов для разработчиков. За вызовы API взимается плата в размере 1 доллар США за 10000 вызовов (т.е. 0,01 цента за запрос), что превышает 1000 в день для бесплатного лимита.

Был взят в рассмотрение погодный проект компании AerisWeather. Основанная в 1996 году, миссия AerisWeather заключалась в том, чтобы «быть самым надежным источником погоды в стране». Компания предоставляет подробную и всеобъемлющую метеорологическую информацию, включая местные прогнозы погоды, расширенные прогнозы и карты погоды.

AerisWeather постоянно работает над расширением своей продуктовой линейки, которая включает Aeris Wear, Aeris Weather API, Aeris Weather Maps и Aeris Web. Airis Wear устройства. Погодные приложения с API-интерфейсом Aeris, показывают региональные и локальные изображения радаров, и в то же время оптимизированы для устройств Android Wear. Существует также бесплатная и профессиональная версия Android-приложения Aeris Wear, которое предлагает покупки в внутри приложения.

Разработчики могут использовать Aeris Weather APITrack для включения метеорологических данных в сторонние приложения для работы в Интернете, также на платформах iOS и Android. Компания предоставляет Aeris JavaScript SDK, который включает в себя множество предустановленных виджетов JavaScript, которые могут быть легко добавлены на веб-сайты или приложения. API Aeris Weather доступен для разработчиков бесплатно; Тем не менее, бесплатный план API ограничен. Существует несколько платных планов, от базового до корпоративного.

В ходе анализа веб-служб вызвало интерес национальное управление по исследованию океанов и атмосферы (NOAA). Оно обеспечивает программный доступ к комплексным наборам данных о погоде через национальную службу погоды NOAA APITrack. НОАА также обеспечивает программный доступ к климатическим данным, геопространственным информационным системам и данным, имеющимся в хранилище Historical Observing Metadata Repository (HOMR). Эти наборы данных предоставляются национальным центром климатических данных NOAA (NCDC), который, согласно веб-сайту, ведет крупнейший в мире архив климатических данных, собирающий данные с наземных станций, судов, буев, радаров, государств.

Не был оставлен без внимания и российский сервис Gismeteo. Он был запущен в 1998 году и очень быстро завоевал популярность. В свое время он являлся лидером по предоставлению достоверных прогнозов для городов России и ближнего зарубежья. Число посетителей в первые десять лет выросло с 300 000 до 6,5 миллионов. На данный момент прогнозы охватывают более 14 000 локаций по всему миру и данное число постоянно растет. Данная погодная служба в наши дни, также, как и остальные, предоставляет программный интерфейс для разработчиков, позволяя иметь прогнозы на дни, недели вперед. Стоит заметить, что данная служба обладает способностью в предоставлении данных на три и более часов вперед.

Каждая компания-поставщик погодных данных предоставляет очень некомфортные условия для разработчиков без стартового капитала. Было принято решение о выборе Accuweather API, Dark Sky API и Aeris Weather, так как данные службы имеют широкий спектр предоставляемых погодных характеристик бесплатно всего за 1000 вызовов в сутки.

**4.2 Анализ спектра предоставляемых данных**

Имея надежные источники погодных данных, было принято решение о детальном рассмотрении ответов, возвращаемых данными погодными веб службами. В ходе анализа были найдены не только измерения температуры воздуха и показатели скорости и направления ветра, но и данные о видимости, атмосферном давлении, температуры по ощущениям, и многое другое, пример json ответа о текущем состоянии в городе Минске на 18 мая 2017 года от веб-службы Accuweather можно увидеть на рисунке 4.1.



Рисунок 4.1 – текущее состояние от AccuWeather

Если обратить внимание на то, как предоставляется температура и давление, то можно приятно удивиться, увидев данные в различных системах счисления. Пример такого результата можно увидеть на рисунке 4.2

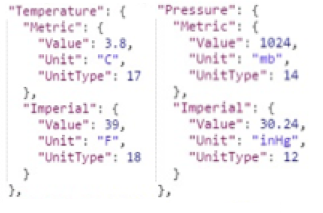


Рисунок 4.2 – разнообразие в представлении данных от AccuWeather

Также не остаются без внимания ссылки на фотографии данного города, в нашем случае - города Минска. Пример одной из фотографий можно увидеть на рисунке 4.3.



Рисунок 4.3 – присутствие ссылки на фотографию в ответе от AccuWeather

Подводя небольшой итог, можно смело сказать, что веб-сервис Accuweather очень хорошо себя зарекомендовал, предоставляя данный диапазон погодных физических величин на бесплатном тарифе доступа.

Служба Dark Sky предоставляет меньшее количество данных о погоде, что отчетливо видно на рисунке 4.4

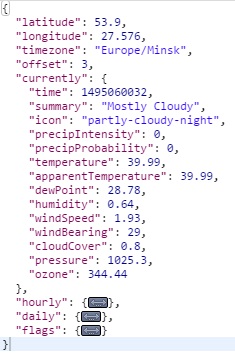


Рисунок 4.4 – ответ о текущем состоянии от Dark Sky

В то же время стоит заметить большой плюс: в ответе от сервиса уже присутствует прогноз по часам вперед, что ослабляет нагрузку на логику со стороны клиента и уменьшает число запросов к самой веб-службе, в то время как каждый запрос стоит денег.

Так как основная задача данной работы состоит в выявлении лидера, путем постоянного мониторинга предоставляемых данных, то было принято решение об анализе температуры и скорости ветра, так как это две наиболее важные величины в прогнозах, на которые в первую очередь обращают внимание пользователи, поэтому эти величины должны обязательно быть предельно точными.

**4.3 Локации и частота выполняемых измерений**

Имея возможность получать прогнозы и текущее состояние окружающей среды за окном в любом месте на планете, привело к вопросу о выборе геопозиций. Так как бесплатный тариф ограничен 1000 запросами от каждой службы, было принято решение о покрытии только городов Республики Беларусь. Все города, вошедшие в рассмотрение изображены на рисунке 4.5.

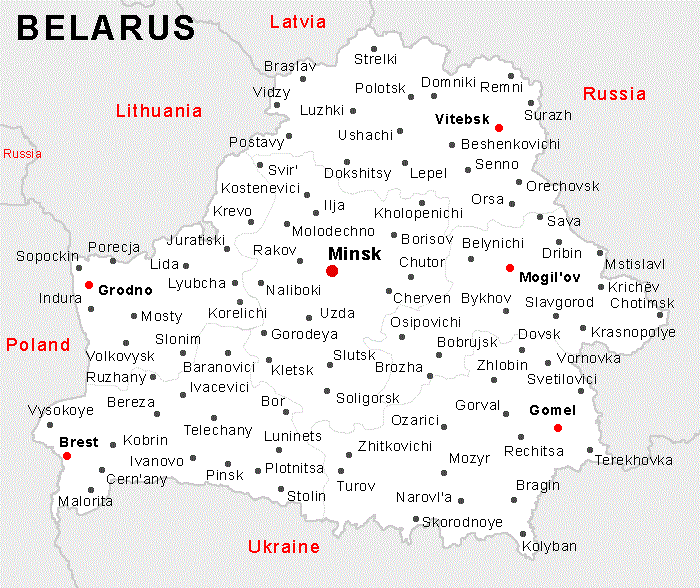


Рисунок 4.5 – города, вошедшие в анализ

Выбранные погодные веб-службы предоставляют широкий спектр прогнозов, начиная от поминутного и заканчивая прогнозами на дни и недели вперед. Рассмотрение прогнозов на день вперед и более дней, и дальнейший анализ не представляется существенным. Результаты в таком случае будут усреднены и не представлять ценности. В то же время поминутный прогноз и анализ с текущим состоянием окружающей среды не может быть достигнут из-за жесткой ограниченности в количестве вызовов в день. В итоге было принято решение о почасовом анализе погодных условий городов Республики Беларусь, что удобно вписывается под возможности данного модуля.

**4.4 Оценивание точности прогнозов**

Основной задачей является выявление лидера среди погодных веб-сервисов. Не имея критериев точности, сделать это не предстоит возможным. Как следствие возникла потребность в определении границ допустимых погрешностей, отклонений от реальных, текущих показателей. Оценивая плавность изменения температуры и скорости ветра в ходе дня, было принято решение об определении оценок точности, которые указаны в таблице 4.1.

Таблица 4.1 – определение правил оценивания точности.

|  |  |  |
| --- | --- | --- |
| Точность показания | Расхождение температуры (С) | Расхождение в скорости ветра (мили/час) |
| Точное | От 0 до 0.5 | От 0 до 1 |
| Близкое к точному | От 0.5 до 1 | От 1 до 2 |
| Неточное | От 1 и более | От 2 и более |

Стоит заметить, что трех уровней точности более, чем достаточно, если брать во внимание, что учет числа результатов "близких к точному" будет только в спорных случаях.

**4.5 Предоставляемые отчеты**

Работающий модуль и одновременно ничего не производящий - пустая трата времени и ресурсов. Так как основной задачей стоит анализ погодных данных и выявление лидера в прогнозах, то наилучшим результатом работы будет таблица с результатами. Таблица - универсальная форма хранения всех видов информации, начиная от дат и наименований, и заканчивая целыми и дробными числами. Имея таблицу в распоряжении, можно без труда с помощью подручных средств сформировать график, диаграмму, гистограмму, схему и многое другое.

Так как объем результатов анализа может быть велик и достигнуть пятисот килобайт или даже больше, и имея в виду то, что количество получателей отчета может вырасти от единиц до десятков, то было решено использовать рассылку на почтовые ящики заинтересованным лицам. Данный способ является наилучшим решением, так как объем письма обычно ограничен мегабайтами данных, что с лихвой удовлетворяет потребности.

Как только пользователь был добавлен в список получателей отчетов, то в тот же момент ему отправляется письмо-приветствие, в котором указано, что он является счастливым обладателем результатов анализа погоды, которую обрабатывает настроенный модуль. Пример письма-приветствия изображен на рисунке 4.6.

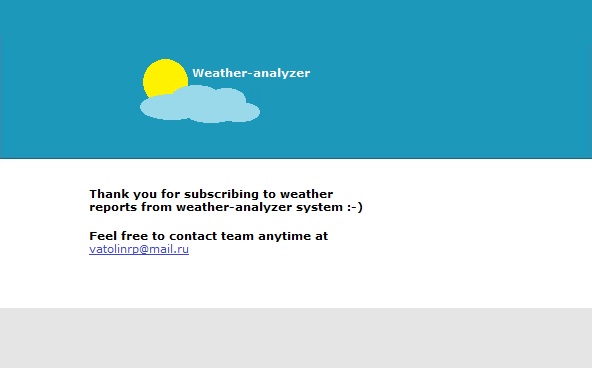


Рисунок 4.6 – сообщение-приветствие

Допустим модуль отработал определенное время и сформировал таблицу. Сразу возникает задача: каким способом эту таблицу передать посредством электронного письма. Среди возможных вариантов, попавших под рассмотрение были: передача в виде текстовой строки, сохранение на стороннем ресурсе и прикрепление защищенной ссылки на таблицу, сохранение в текстовом файле и прикрепление к письму, сохранение в определенном формате и прикрепление к письму. Так как наиболее распространенной программой для работы с таблицами является Microsoft Exel, то было принято решение о записи результата работы в файл, с расширением CSV.

CSV-файлы (от английского Comma Separated Values - значения, разделенные запятыми) - текстовый формат, специально разработанный для представления табличных данных.

Все валидные CSV файлы подчиняются следующей спецификации:

1. Каждая строка файла – это одна строка таблицы;
2. Разделителем значений колонок является символ запятой;
3. Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("). Если в значении встречаются кавычки - они представляются в виде двух кавычек подряд.

Пример, взятый с ресурса wikipedia.org четко и понятно иллюстрирует использование данного формата. Пусть текстовой файл заполнен данными, изображенными на рисунке 4.7.

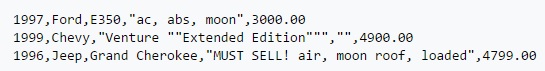


Рисунок 4.7 – пример содержания CSV файла.

Данный текстовой файл будет без труда открыт с помощью программы Microsoft Excel, где мы увидим уже сформированную таблицу. В представленном случае это будет выглядеть, как показано на рисунке 4.8.

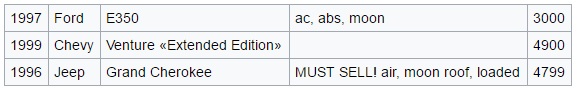


Рисунок 4.8 – отображение CSV файла в Microsoft Excel.

Выбранный вариант передачи информации очень удобен и дешев в использовании, что является очень правильным решением в данной работе.

В результате заинтересованному лицу придет сообщение на электронную почту, в котором он увидит данный файл прикрепленным. На рисунке 4.9 представлен пример письма-отчета.

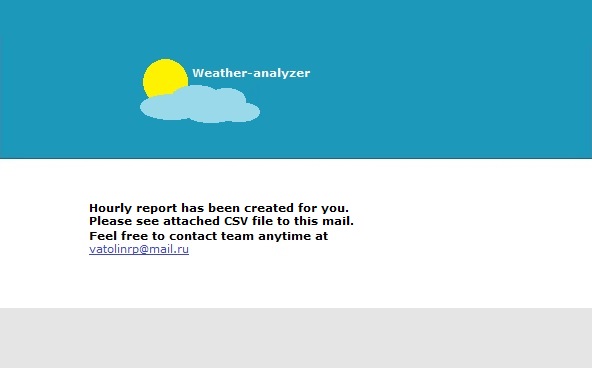


Рисунок 4.9 – письмо-отчет у пользователя системы.

Представленная рассылка писем очень удобна как для обычных пользователей, так и для научных сотрудников.

**РЕЗУЛЬТАТЫ И АНАЛИЗ**

**5.1 Рассмотрение полученных писем-отчетов**

В течение недель модуль был в непрерывном штатном рабочем режиме и выполнял обработку данных, генерируя и отправляя отчеты каждый час. Пример полученной таблицы для города Минск в период времени в день от веб-службы Accuweather можно увидеть на рисунке 5.1.

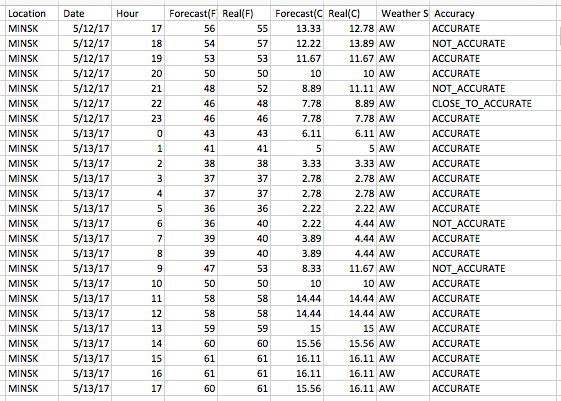
****

Рисунок 5.1 – почасовой отчет по городу Минск от Accuweather

В то же время для города Брест можно рассмотреть на рисунке 5.2

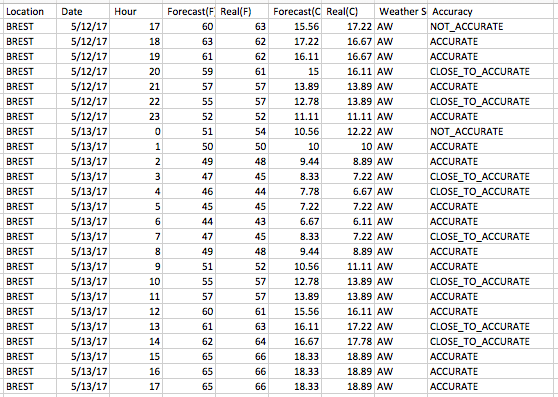


Рисунок 5.2 – почасовой отчет по городу Брест от Accuweather

Как видно из таблиц выше, с данным представлением очень удобно в дальнейшем работать.

Добавим в рассмотрение еще таблицы с этими городами и сделаем по этим четверым графики (рисунки 5.3-5.6).

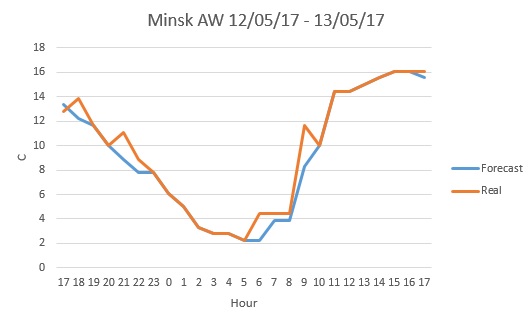


Рисунок 5.3 – график почасового отчета по городу Минск от Accuweather

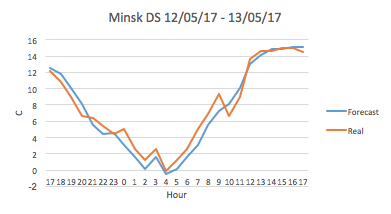


Рисунок 5.4 – график почасового отчета по городу Минск от Dark Sky

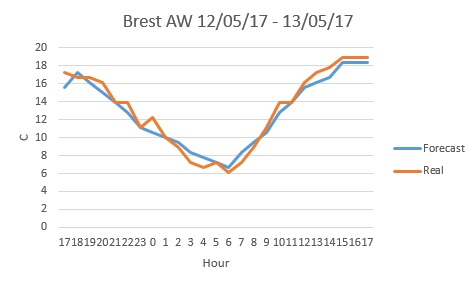


Рисунок 5.5 – график почасового отчета по городу Брест от Accuweather

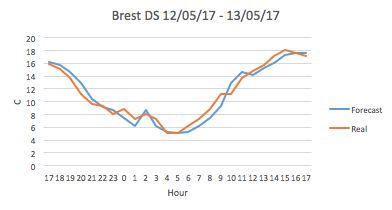


Рисунок 5.6 – график почасового отчета по городу Брест от Dark Sky

Как можно заметить, погрешность есть и в основном она в градус Цельсия. Рассмотрим по аналогии графики по скорости ветра. Данные по ветру представлены на рисунках 5.7-5.10.

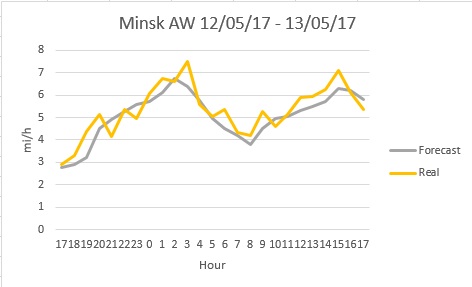


Рисунок 5.7 – график почасового отчета по городу Минск от Accuweather

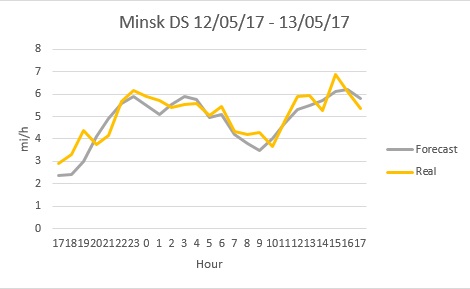


Рисунок 5.8 – график почасового отчета по городу Минск от Dark Sky

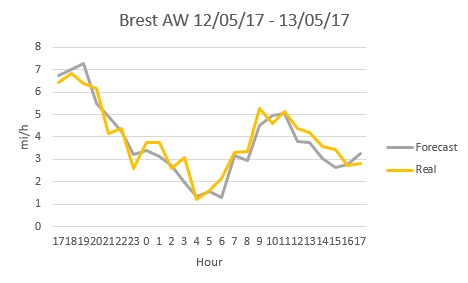


Рисунок 5.9 – график почасового отчета по городу Брест от Accuweather

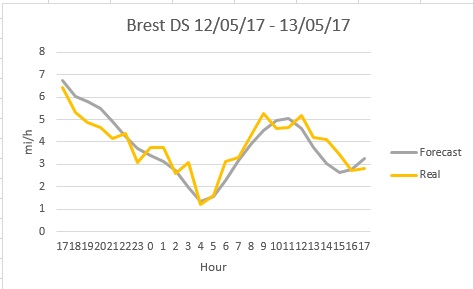


Рисунок 5.10 – график почасового отчета по городу Брест от Dark Sky

По данным графикам можно сделать вывод, что и точность определения скорости ветра у этих погодных веб-служб примерно одинаковая.

Подводя итог можно вполне уверенно сказать, что погрешность есть и она совсем незначительная. Не рассматривая результаты работы других сервисов, можно с уверенностью сказать, что результаты их прогнозов будут весьма похожи.

**5.2 Рассмотрение веб-служб по локациям**

Вместе с общей таблицей, представленной выше, в почасовом отчете присутствует и дополнительная таблица, которая говорит о том, какая погодная веб-служба гарантировала за последнее время наибольшее число точных прогнозов. Отрывок данной таблицы изображен на рисунке 5.11.



Рисунок 5.11 – отрывок таблицы с результатами точных прогнозов от служб

Данные города Республики Беларусь находятся в разных точках страны, откуда можно сделать вывод о том, что служба AccuWeather достаточно точно работает на всей территории государства.

**ЗАКЛЮЧЕНИЕ**

В данной работе:

* применены навыки разработки приложений на Java;
* изучен фреймворк Apache Storm и взаимодействие его компонентов;
* рассмотрен спектр погодных веб-служб, предоставляющих бесплатный доступ к метеоданным;
* изучено и применено взаимодействие со сторонними веб-службами по протоколу http;
* изучены библиотеки для преобразования Java объектов из файлов типа JSON;
* изучен и использован формат файлов CSV;
* изучены Java библиотеки для отправки писем на электронные почтовые ящики;
* изучен и интегрирован кэш Ehcache в разработанный модуль;
* применены знания Microsoft Excel при построении графиков и организации данных в таблице;
* применены знания о паттернах проектирования в ходе разработки модуля.
* использовано юнит-тестирование, для проверки вычислений отклонений от точности.

Таким образом, разработанный модуль позволяет не только предоставить почасовые отчеты о погодных данных, но еще и выявить лидера среди точных прогнозов по определенному критерию, будь то температура воздуха, атмосферное давление и скорость ветра. Испытание модуля показало наилучшие результаты у погодной веб-службы Accuweather для городов Республики Беларусь. Были выбраны для сравнения такие показатели как: температура воздуха и скорость ветра. У Accuweather было замечено в среднем отклонение от точности до 2 градусов максимум и до 2 мил в час для скорости ветра.

Разработанный модуль достойно выполнил свою задачу, в которой выявил безусловного лидера.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Эккель, Б. Философия Java. Библиотека программиста / Б. Эккель. – СПб: Питер, 2001. – 880 с.;
2. Буч, Г. Объектно-ориентированный анализ и проектирование / Г. Буч. – Москва: 2-е изд. Пер. с англ. – М.: «Издательство Бином», 1998 – 560с.;
3. Хабибуллин, И. Ш. Создание распределённых приложений на Java 2 / И. Ш. Хабибуллин – СПб: БХВ-Петербург, 2001 г. – 688с.;
4. Блинов, И. Н. Java. Промышленное программирование: практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск: УниверсалПресс, 2007. – 704 с;
5. Гранд, М. Шаблоны проектирования в Java / М. Гранд – Москва: Пер. с англ. С. Беликовой. – М.: Новое знание, 2004. – 559 с.;
6. Habrahabr [Электрон. ресурс] / Habrahabr Project – Москва, 2017. - Режим доступа: https://habrahabr.ru/post/186208 – Дата доступа: 16.05.2017;
7. Habrahabr [Электрон. ресурс] / Habrahabr Project – Москва, 2017. - Режим доступа: https://habrahabr.ru/company/mailru/blog/129476 – Дата доступа: 02.03.2017;
8. Ehcache [Электрон. ресурс] / Ehcache Project – Atlanta, 2017. - Режим доступа: http://www.ehcache.org/documentation/3.3/caching-terms.html – Дата доступа: 21.04.2017;
9. JSON [Электрон. ресурс] / JSON Data Interchange – Mосква, 2013. - Режим доступа: http://www.json.org/json-ru.html – Дата доступа: 11.05.2017.