**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, \_\_ с., \_\_ рис., \_ табл., \_ источника.

*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](#_Toc474100881)

[**1.** **STORM FRAMEWORK** 5](#_Toc474100882)

[1.1 Ключевые особенности фреймворка 5](#_Toc474100883)

[1.1 Элементы Storm 5](#_Toc474100884)

[1.1 Сохранность данных 6](#_Toc474100885)

[1.1.1 На уровне Spout 6](#_Toc474100886)

[1.1.1 На уровне Bolt 6](#_Toc474100887)

[1.1.1 На уровне Tuple 7](#_Toc474100888)

## 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.
   1. **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.

1. **На уровне Spout**

Если нам не важно были ли ошибки при обработке Tuple, то Spout отправляет Tuple в SpoutOutputCollector посредством вызова метода:

|  |  |
| --- | --- |
|  | (1) |

Eсли необходимо узнать успешно ли обработался Tuple, то вызов будет выглядеть как:

|  |  |
| --- | --- |
|  | (2) |

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.

1. **На уровне Bolt**

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, (5) вызывается после выхода из метода execute автоматически.

1. **На уровне Tuple**

При обработке входного Tuple, Bolt может генерировать более одного выходного Tuple. Если Bolt вызвал 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**