МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

На правах рукописи

П.Д. Кравченя, А.Е. Андреев

Системы обработки больших данных

Учебное пособие

Волгоград
2021

УДК 004.45:004.056

Кравченя П.Д.

Системы обработки больших данных: учеб. пособие / П.Д. Кравченя, А.Е. Андреев ВолгГТУ. – Волгоград, 2021. – 40 с.

В учебном пособии рассмотрены методы и технологии работы с большими данными, задания к практическим работам и лабораторный практикум.

Учебное пособие предназначено для магистров, обучающихся по программам магистратуры по профилю «искусственный интеллект» по направлениям 09.04.01 «Информатика и вычислительная техника», 09.04.03 «Прикладная информатика», 09.04.02 «Информационные системы и технологии». Учебное пособие выполнено в рамках реализации гранта на разработку программ бакалавриата и программ магистратуры по профилю «Искусственный интеллект», а также на повышение квалификации педагогических работников образовательных организаций высшего образования в сфере искусственного интеллекта (конкурс 2021-ИИ-01 от 10.06.2021).

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основные требования к работе с большими данными такие же, как и к любым другим наборам данных. Однако массовые масштабы, скорость обработки и характеристики данных, которые встречаются на каждом этапе процесса, представляют серьезные новые проблемы при разработке средств. Целью большинства систем больших данных является понимание и связь с большими объемами разнородных данных, что было бы невозможно при использовании обычных методов.

Исключительный масштаб обрабатываемой информации помогает определить системы больших данных. Эти наборы данных могут быть на порядки больше, чем традиционные наборы, что требует большего внимания на каждом этапе обработки и хранения.

Поскольку требования превышают возможности одного компьютера, часто возникает проблема объединения, распределения и координации ресурсов из групп компьютеров. Кластерное управление и алгоритмы, способные разбивать задачи на более мелкие части, становятся в этой области все более важными. Данные часто поступают в систему из нескольких источников и должны обрабатываться в режиме реального времени, чтобы обновить текущее состояние системы.

Существует несколько различных подходов в обработке больших данных, но в стратегиях и программном обеспечении есть общие черты.
Основные категории, связанные с обработкой больших данных:

- Внесение данных в систему;
- Сохранение данных в хранилище;
- Вычисление и анализ данных;
- Визуализация результатов.

1. ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

В таблице представлен образец датасета, представляющий собой фрагмент лог-файла, фиксирующего события запуска (**started**) и остановки (как успешной (**finished**), так и неуспешной (**failed**)) системных служб некоторой UNIX-системы.

Дата	**Время**	**Служба**	**Статус**
2020-10-03	13:25	sshd	started
2020-12-12	10:01	ftpd	finished
2020-12-12	11:45	ftpd	failed
2020-12-12	12:14	ftpd	failed
2020-12-12	09:23	sshd	failed
...			
2020-12-08	00:04	httpd	failed

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите количество дат, за которые служба sshd завершила некорректно свою работу большее количество раз, чем служба ftpd.

Оформите его в виде псевдокода, показав реализацию основных операций над данными.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Требуется ли в данной задаче для повышения производительности создавать специальную реализацию класса *Combiner*? Если да, то каким образом? Если нет, то почему? Поясните ответы.

Решение:

а, б) В соответствии с принципами работы MapReduce в системе Hadoop, представим алгоритм решения задачи в виде последовательности заданий (jobs), каждое из которых включает чтение требуемых данных из HDFS, обработку их с помощью одной операции MapReduce и последующую запись полученных результатов обратно в HDFS.

Job 1:

В качестве класса InputFormat для первого job’а логичнее всего выбрать TextInputFormat, поскольку входной файл представлен в виде текста с несколькими записями. Данный класс читает входной файл и формирует набор пар <ключ, значение>, которые передаются на обработку этапу map. Входным ключом выступает позиция первого символа строки в файле, а значением – содержимое строки.

Поскольку, согласно заданию, в каждую дату любая из системных служб может некорректно завершить свою работу любое количество раз (в различные моменты времени), в первом job’е логично подсчитать количество failed-статусов для каждой службы в каждую дату. Для этого на этапе Map необходимо для каждой записи, содержащей информацию о некорректном завершении sshd или ftpd, сформировать ключ в виде кортежа (служба, дата), а в качестве значения поставить единицу. Записи о службах, отличных от sshd и ftpd, а также записи со статусами started и finished для решения задачи не требуются. Поэтому, их можно не учитывать при формировании данных для следующего этапа.

Преобразование данных (ключей и значений) на первом этапе Map можно представить следующим образом:

Map 1: <идентификатор первого символа в строке, строка текста> → <(служба, дата), 1>.

Шаблонный класс можно в этом случае определить так:

class Mapper1 extends Mapper<Object, Text, WritableComparable[], IntWritable>

Здесь ключ представлен типом Object, поскольку он не используется в функции. Значение (строка текста) имеет тип Text. Формируемые ключи представляют собой кортеж, здесь они представлены массивом WritableComparable[] (конкретная реализация класса для этого кортежа может быть выполнена на усмотрение программиста, но класс должен наследоваться от WritableComparable, заимствуя у него необходимые для ключей методы). Формируемое значение – единица для всех записей – представлено типом IntWritable.

Прототип и реализация в виде псевдокода функции map() первого job'а могут быть записаны следующим образом:

```
public void map(Object key, Text value, Context context) {
   распарсить строку value, получить службу, дату и статус;
      если служба = sshd или ftpd, а статус = failed, сформировать результат этапа map и записать его в переменную контекста:
      context.write( [служба, дата], 1);
}
```

На этапе Reduce первого job'а нужно просуммировать значения всех пар <key, value>, сформированных на этапе Map, с одинаковыми значениями ключей. Преобразование данных можно представить таким образом:

Reduce 1: <(служба, дата), [1, 1, 1 … 1]> → <(служба, дата), N>.

Шаблонный класс определяется следующим образом:

class Reducer1 extends Reducer<WritableComparable[], IntWritable, WritableComparable[], IntWritable>

Типы ключей и значений на входе Reduce совпадают с типами данных, сформированных предыдущим этапом Map. Типы выходных ключей и значений не отличаются от входных.

Прототип и реализация данного этапа Reduce может быть записана в следующем виде. На вход поступают ключ и контейнер со всеми значениями, ему соответствующими:

void reduce(WritableComparable[] key, Iterable<IntWritable> values, Context context) {

просуммировать все значения в values и записать результат в переменную контекста:

context.write(key, сумма);

}

Таким образом, выполнение первого job'а позволяет получить количество некорректных завершений работы служб ftp и sshd в каждую дату, указанную во входном файле.

Job 2:

Зная количество некорректных остановок обеих служб в каждую дату, можно определить, в какие даты для какой службы их было больше. Для этого воспользуемся следующим приемом. На этапе Map каждой входной паре <(служба, дата), N> поставим в соответствие либо пару <дата, +N>, если в качестве службы указан sshd, либо пару <дата, -N>, если службой является ftpd.

Для чтения входной информации из HDFS при выполнении второго job'а рационально использовать класс KeyValueTextInputFormat, потому что данные, полученные первым job'ом, представляют собой набор ключей и значений с разделителем между ними, представленных в текстовом виде.

Преобразование данных (ключей и значений) на втором этапе Map тогда запишется в виде:

Map 2: <(служба, дата), N> → <дата, ±N>.

А шаблонный класс, соответствующий операции Map, примет вид:

class Mapper2 extends Mapper<Text, Text, Text, IntWritable>

Входные ключи и значения формируются в текстовом виде классом InputFormat, а выходные ключи и значения являются текстовыми и целочисленными соответственно.

Прототип и реализация в виде псевдокода функции map() второго job'а запишется следующим образом:

```
public void map(Text key, Text value, Context context) {
   распарсить строку key и получить службу и дату;
   преобразовать value в целочисленный тип;
            если служба = sshd, записать в переменную контекста положительное значение value, а если служба = ftpd, то отрицательное. В качестве ключа оставить только дату:
      context.write( дата, ±value);
}
```

Сформированные этапом Map данные включают не более двух записей с одним и тем же ключом (представляющим конкретную дату) и с отрицательным и/или положительным числом в качестве значения (любая из них или обе могут отсутствовать, если в соответствующую дату некорректной остановки какой-либо из рассматриваемых служб не было). Теперь на этапе Reduce необходимо просуммировать эти значения для одинаковых ключей. Если сумма окажется положительной, то в рассматриваемую дату служба sshd аварийно останавливалась большее количество раз, чем ftp, а если отрицательная – то служба ftpd. Нулевое

значение соответствует равному количеству некорректных завершений работы сервисов.

Таким образом, преобразование ключей и значений при выполнении второго Reduce примет вид:

Reduce 2: <дата, [±N, ±N]> → <дата, Σ>.

Шаблонный класс при этом может быть представлен следующим образом:

class Reducer2 extends Reducer<Text, IntWritable, Text, IntWritable>

А прототип и реализация функции reduce():

void reduce(Text key, Iterable<IntWritable> values, Context context) {

просуммировать все значения в values и записать результат в переменную контекста:

context.write(key, сумма);

}

В результате выполнения второго задания (job) удалось определить, какая из служб некорректно останавливалась большее количество раз в определенную дату.

Job 3:

Третье задание (job) позволяет решить задачу. Зная для каждой даты службу с наибольшим числом аварийных остановок, можно подсчитать количество таких дат. Для этого на этапе Map для каждой записи с положительным значением (соответствующей факту того, что в данную дату сервис sshd некорректно завершал свою работу большее число раз, чем ftpd) сформируем пару с любым ключом (например, равным единице) и единичным значением:

Map 3: <дата, Σ> → <1, 1>.

Шаблонный класс для Map в этом случае запишется так:

class Mapper3 extends Mapper<Object, Text, IntWritable, IntWritable>

Тип данных ключа был изменен с текстового на целочисленный, поскольку вычислительным системам проще оперировать целыми числами, чем строками.

public void map(Object key, Text value, Context context) {

преобразовать в целочисленный тип value;

если value > 0, то записать в переменную контекста единицу в качестве как ключа, так и значения:

context.write(1, 1);

}

На выходе этапа Map сформировано множество записей с одинаковым значением ключа и единичными значениями. Осталось подсчитать их количество:

Reduce 3: <”A”, [1, 1, 1 … 1]> → <”A”, M>.

class Reducer3 extends Reducer< IntWritable, IntWritable, IntWritable, IntWritable>

void reduce(IntWritable key, Iterable<IntWritable> values, Context context) {

просуммировать все значения в values и записать результат в переменную контекста:

context.write(key, сумма);

}

В результате выполнения всех трех job’ов в HDFS окажется файл с единственной записью вида «ключ:значение», которая и будет ответом на поставленную задачу.

в) Как известно, с целью сокращения объема передаваемых между узлами данных в Hadoop-приложениях запускают этап Combine. Данный этап выполняет над локальными данными операции, аналогичные операциям этапа Reduce, и в некоторых случаях может использовать ту же функцию reduce(). Однако, количество запусков этапа Combine заранее не определено, поэтому он может использовать только такие функции, многократный запуск которых не приводит к неверному результату вычислений. Это требование, кстати, включает и то, что типы данных ключей и значений в используемой функции изменяться не должны.

Если функция reduce() не удовлетворяет этим требованиям, то для этапа Combine приходится разрабатывать другую функцию. Иногда, даже при «подходящей» функции reduce(), для этапа Combine разрабатывается специальная функция. Часто это делается для повышения производительности вычислений с учетом особенностей локального исполнения Combine.

Как можно видеть из вышеприведенного решения задачи, все используемые в нем функции reduce() удовлетворяют представленным требованиям. Они выполняют одинаковую операцию: суммирование всех элементов с одинаковым значением ключа. При этом довольно сложно предложить способ увеличения производительности выполнения этой операции, принимая во внимание локальное расположение всех суммируемых элементов на конкретном узле. Поэтому, можно сказать, что разработка специальной реализации функции для этапа Combine в данной задаче не требуется.

2. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

2.1 Лабораторная работа №1. Изучение технологий Hadoop и MapReduce

Цели работы:

1. Получить представление о технологии MapReduce;
2. Познакомиться с технологией Hadoop, обеспечивающей хранение и обработку больших объемов данных на кластерных системах;
3. Получить представление о сущности и принципах работы распределенной файловой системы Hadoop (HDFS) и системы планирования заданий и управления Hadoop-кластером;
4. Получить навыки разработки простых программ на основе технологии MapReduce.

2.1.1 Теоретические основы

К категории «**Большие данные** (*Big Data*)» относится информация, которую из-за ее объема уже невозможно обрабатывать традиционными способами. Они представляют собой наборы данных, размер которых достигает такого предела, что ими становится тяжело оперировать, используя имеющиеся инструменты управления данными. Трудоёмкими становятся сбор и хранение, распределение, анализ и визуализация данных. В современном мире объемы больших данных постоянно растут, стремясь от нескольких десятков терабайтов к петабайтам данных.

Для эффективной обработки больших объёмов данных при приемлемых временных затратах необходимы особые технологии. Такими технологиями можно считать глубинный анализ данных (data mining), технологию Apache Hadoop Framework, распределенные файловые системы, распределённые базы данных, алгоритмы MapReduce и т.д.

MapReduce — модель распределённых вычислений, представленная компанией Google и используемая для параллельных вычислений над очень большими наборами данных в компьютерных кластерах. В рамках данной модели приложение представляется в виде серий задач map и reduce. *Задачи map* вызывают функции map для обработки наборов входных данных. Затем *задачи reduce* вызывают функции reduce для обработки промежуточных данных, сгенерированных функциями map, формируя окончательные выходные данные. Задачи map и reduce выполняются изолированно друг от друга, что обеспечивает параллельность и отказоустойчивость вычислений.

Преимущество MapReduce заключается в возможности распределенной обработки данных. Операции map работают параллельно и независимо друг от друга. Аналогично, множество рабочих узлов могут осуществлять операцию reduce — для этого необходимо, чтобы все результаты этапа map с одним конкретным значением ключа обрабатывались одним рабочим узлом в один момент времени. Хотя этот процесс может быть менее эффективным по сравнению с более последовательными алгоритмами, MapReduce может быть применен к большим объёмам данных, которые могут обрабатываться большим количеством серверов.

Hadoop — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов. Проект Hadoop разработан на языке Java в рамках вычислительной парадигмы MapReduce. Apache Hadoop состоит из:

- *Hadoop Common*: библиотек и сценариев управления распределенной обработкой, файловой системой, развертывания

инфраструктуры;

- *Hadoop Distributed File System*: распределенной файловой системой, которая обеспечивает высокоскоростной доступ к данным приложения;
- *Hadoop YARN*: системы управления распределёнными приложениями.

2.1.2 Порядок выполнения лабораторной работы

1. Установите соединение с кластером и откройте удаленный рабочий стол. Номер учетной записи выберите в соответствии с номером рабочего компьютера в аудитории. Логин и пароль к учетной записи получите у преподавателя.
2. Исследуйте веб-интерфейс Cloudera Manager, доступный по адресу **http://localhost:7180**, позволяющий управлять Hadoop-кластером и получать информацию о системе в удобном графическом представлении (см. рисунок 1). Проанализируйте представленную информацию.
3. Создайте в домашней директории папку, в которой будут располагаться Ваши файлы. В имя директории желательно включить фамилии пользователей:

$ mkdir ~/ivanov_directory

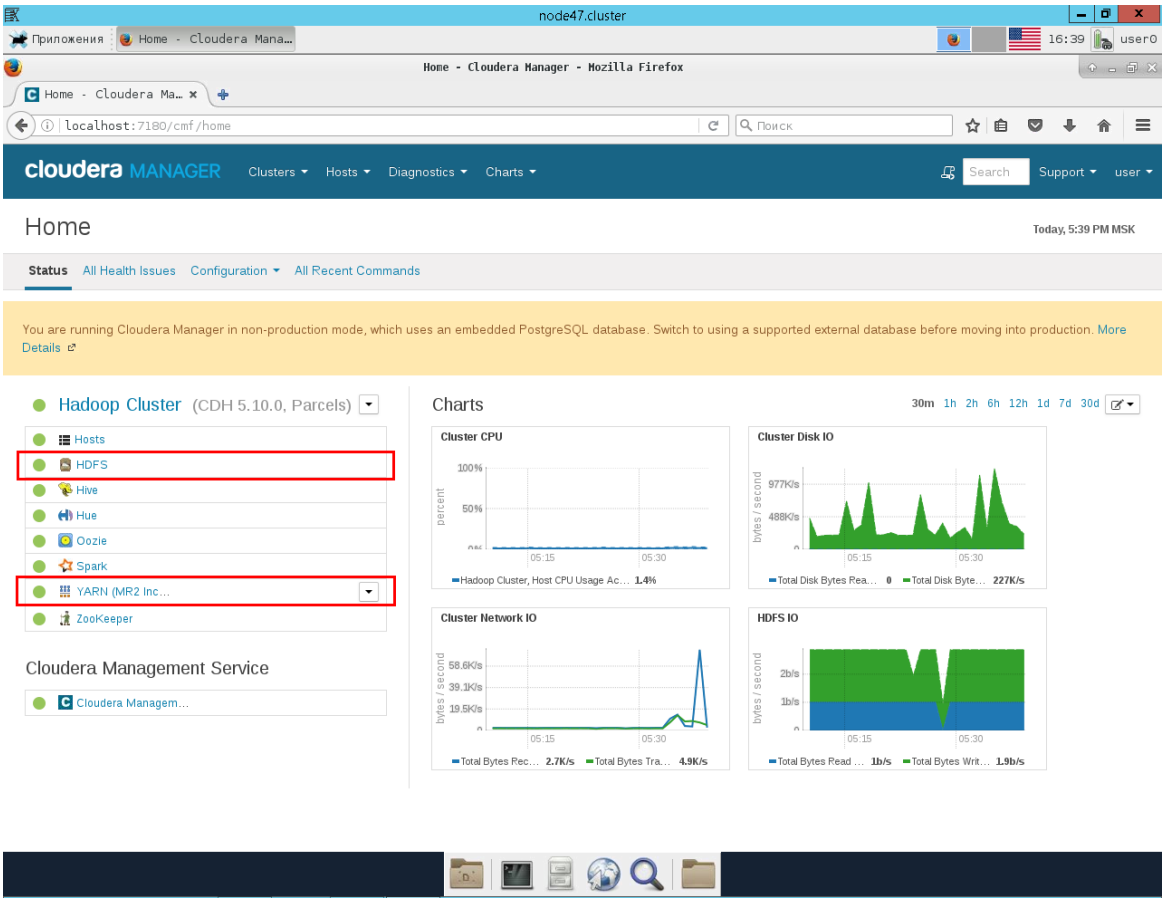

Рисунок 1. Внешний вид интерфейса Cloudera Manager

4. В созданной директории создайте еще одну папку:

$ **mkdir ~/ivanov_directory/WordCount**

5. Скопируйте файл с исходным кодом *WordCount.java* и набор текстовых файлов в созданную директорию *WordCount* с помощью программы WinSCP.

6. Узнайте объем свободного пространства в файловой системе HDFS и выведите содержимое Вашей домашней папки в ней:

$ **hadoop fs -df -h**

$ **hadoop fs -ls /user/user0**

7. Создайте в системе HDFS папку, аналогичную созданной ранее:

$ **hadoop fs -mkdir -p /user/user[0-9]/ivanov_directory/WordCount**

8. Аналогичным образом создайте в системе HDFS директорию, необходимую для хранения исследуемых текстовых файлов:

$ **hadoop fs -mkdir /user/user[0-9]/ivanov_directory/WordCount/input**

9. Скопируйте полученные у преподавателя текстовые файлы в папку *input*. Проверьте их наличие после копирования:

$ **hadoop fs -copyFromLocal ~/ivanov_directory/WordCount/*.txt /user/user[0-9]/ivanov_directory/WordCount/input**

$ hadoop fs -ls /user/user[0-9]/ivanov_directory/WordCount/input

10. Установите требуемые для работы переменные окружения:

$ **export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera/**

$ **export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:/ $JAVA_HOME/lib/tools.jar**

11. Перейдите в директорию *WordCount*. Скомпилируйте файл с исходным кодом *WordCount.java*, который представляет программу, определяющую количество повторений слов в наборе текстовых файлов. Для этого в командной строке выполните следующую команду:

$ **hadoop com.sun.tools.javac.Main WordCount.java**

12. Соберите Java-пакет из всех полученных в результате компиляции файлов:

$ **$JAVA_HOME/bin/jar cf wc.jar WordCount*.class**

13. Запустите программу для подсчета количества слов в системе Hadoop:

$ **hadoop jar wc.jar WordCount /user/user[0-9]/ivanov_directory/ WordCount/input /user/user[0-9]/ivanov_directory/WordCount/output**

Если система сообщает о том, что директория вывода уже создана, необходимо удалить ее командой:

$ **hadoop fs -rm -r /user/user0/ivanov_directory/WordCount/output**

и запустить программу еще раз.

14. По окончанию работы программы проверьте статус ее завершения. Если выполнение программы произошло с ошибками, исправьте их и запустите расчет вновь.

15. Добейтесь успешного завершения программы. Зафиксируйте и проанализируйте консольный вывод приложения. Просмотрите информацию, предоставляемую Cloudera Manager о завершившейся задаче.

16. Проверьте содержимое директории output. Отметьте формат файлов, находящихся в ней.

$ **hadoop fs -ls /user/user[0-9]/ivanov_directory/WordCount/output**

17. Выведите содержимое всех файлов в директории output в файл result.txt в текущей директории.

$ **hadoop fs -getmerge /user/user0/ivanov_directory/WordCount/ output/ result.txt**

18. Просмотрите содержание полученного файла с помощью текстового редактора и проанализируйте его содержимое. Фрагмент файла поместите в протокол.

19. Доработайте программу по подсчету числа вхождения слов в текстовые файлы, добавив в нее еще один MapReduce этап. Данный этап должен принимать на вход результат подсчета количества вхождений слов в тексты, осуществленный программой WordCount, и выводить слова с характеристикой встречаемости их в тексте (согласно таблице 1):

Таблица 1. Соотношение между количеством слов и характеристикой встречаемости

Количество вхождений слова в тексты	Характеристика встречаемости
1 – 10	Очень редко
10 – 50	Редко
50 – 200	Часто
> 200	Очень часто

20. Проанализируйте результаты, выводимые модифицированной программой. Доработанный исходный код программы, результаты ее работы, скрины информационных окон Cloudera Manager поместите в протокол. Сделайте выводы.

2.1.3 Домашнее задание

Разработайте по вариантам алгоритм и программу с использованием технологии MapReduce на базе Hadoop Framework. Расчет требуется организовать в виде последовательности map/reduce-этапов, позволяющих получить итоговый результат. Выполните для разработанной программы пункты 9–20 и занесите результаты в протокол. Сделайте выводы.

Варианты заданий:

1. На рисунке представлен фрагмент датасета, представляющий собой запись усредненных за сутки показаний метеорологических датчиков.

Дата	**Город**	**Температура**	**Влажность**
20.11.2018	Волгоград	+4	81
13.01.2008	Москва	-17	98
...			
04.05.2015	Мурманск	-10	91

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите последнюю дату, в которую влажность в Волгограде превысила влажность в Москве.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

2. На рисунке представлен фрагмент датасета, представляющий собой запись усредненных за сутки показаний метеорологических датчиков.

Дата	**Город**	**Температура**	**Влажность**
20.11.2018	Волгоград	+4	81
13.01.2008	Москва	-17	98
...			
04.05.2015	Мурманск	-10	91

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите минимальную влажность в городе, в котором интервал температур в марте 2016 года максимальный.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

3. На рисунке представлен фрагмент датасета, представляющий собой запись усредненных за сутки показаний метеорологических датчиков.

Дата	Город	Температура	Влажность
20.11.2018	Волгоград	+4	81
13.01.2008	Москва	-17	98
...			
04.05.2015	Мурманск	-10	91

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите город, в котором интервал температур в апреле 2008 года минимальный.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

4. На рисунке представлен фрагмент датасета с информацией об успеваемости учеников некоторой школы.

Ученик	Класс	Предмет	Оценка
Иванов Петр	7Б	Математика	4
Сидоров Василий	7Б	Физкультура	5
...			
Игнатов Сергей	8А	История	3

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите количество учеников восьмых классов, имеющих среднюю оценку по предметам ниже четверки.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

5. На рисунке представлен фрагмент датасета с информацией об успеваемости учеников некоторой школы.

Дата	**Город**	**Температура**	**Влажность**
Иванов Петр	7Б	Математика	4
Сидоров Василий	7Б	Физкультура	5
...			
Игнатов Сергей	8А	История	3

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите класс(-ы), в котором(-ых) больше всего учеников, имеющих пятерки по физкультуре.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

5. На рисунке представлен фрагмент датасета, представляющий собой информацию о товарах на складе некоторого магазина.

Товар	**Кол-во товара**	**Цена товара**	**Категория товара**

Куртка	19	6800	Одежда
Брюки	34	1500	Одежда
...			
Кресло	8	8400	Мебель

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите выручку магазина, если он продаст все самые дорогие товары в каждой категории.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

7. На рисунке представлен фрагмент датасета, представляющий собой информацию о товарах на складе некоторого магазина.

Товар	**Кол-во товара**	**Цена товара**	**Категория товара**
Куртка	19	6800	Одежда
Брюки	34	1500	Одежда
...			
Кресло	8	8400	Мебель

а) Составьте алгоритм MapReduce, позволяющий на основе данных, представленных выше, решить следующую задачу:

Определите товар с максимальной ценой среди самых многочисленных товаров в каждой категории.

б) Определите классы – параметры шаблонных классов Mapper и Reducer. Напишите прототипы функций map(…) и reduce(…).

в) Увеличит ли производительность использование в данной задаче стандартного варианта класса *Combiner*? Имеет ли смысл изменить его стандартную реализацию? Если да, то каким образом? Поясните ответы.

2.1.4 Теоретические вопросы к отчету лабораторной работы

1. Понятие больших данных (Big Data). Проблемы, связанные с обработкой больших данных традиционными инструментами. Необходимость обработки больших данных.
2. Понятие технологии MapReduce, основные принципы, лежащие в ее основе. Характеристики этапов map и reduce. Примеры задач MapReduce.
3. Hadoop как распределенная система обработки больших данных. Стек технологий Hadoop. Масштабируемость и производительность Hadoop. Принцип определения доступности узлов. Понятие Heartbeat.
4. Понятие HDFS. Архитектура HDFS. NameNode и DataNode: понятие и функции. Обеспечение надежности хранения данных. Понятие репликации. Характеристики и ключевые особенности HDFS как файловой системы.
5. Процесс чтения и записи данных в HDFS.
6. Понятие YARN. Архитектура YARN. ResourseManager и NoneManager: понятие и функции. Понятие контейнеров (container) YARN. Отличия MapReduce v1 от MapReduce v2.
7. Процесс запуска и исполнения приложения под управлением YARN. Понятие ApplicationMaster. Взаимодействие компонентов YARN. Понятие локальности данных.
8. Основные планировщики YARN: FIFO, Capacity, Fair. Их назначение и ключевые особенности.

9. Процесс выполнения приложения в Hadoop MapReduce. Понятие задания (job) и задачи (task).

10. Основные этапы исполнения приложения MapReduce в Hadoop: map, reduce, sort, shuffle, combine. Их назначение и особенности исполнения.

11. Обмен данными между основными этапами в MapReduce Hadoop. Понятие сериализации. Интерфейсы Writable и WritableComparable.

2.2 Лабораторная работа №2. Изучение технологии Apache Spark

Цели работы:

1. Познакомиться с фреймворком Apache Spark, обеспечивающим высокопроизводительную обработку больших данных на кластерных системах;
2. Познакомиться с архитектурой системы Spark и принципами разработки для нее программного обеспечения;
3. Изучить особенности реализации концепции MapReduce для системы Spark;
4. Получить навыки разработки простых программ с использованием Apache Spark.

2.2.1 Теоретические основы

Хотя наиболее широко используемой платформой в области распределенного анализа данных является *Hadoop*, существуют альтернативы, предоставляющие по сравнению с ней некоторые важные преимущества. Одной из таких альтернатив, набирающих популярность в

последнее время, является ***Apache Spark*** — масштабируемая платформа анализа данных. В отличие от классического обработчика из ядра Hadoop, реализующего двухуровневую концепцию *MapReduce* с дисковым хранилищем, система *Spark* использует специализированные примитивы для обработки данных в оперативной памяти, благодаря чему позволяет получать значительный выигрыш в скорости работы. Наиболее значительным он оказывается для приложений, в которых рабочий набор данных многократно используется в параллельных операциях (например, в алгоритмах машинного обучения).

Проект *Spark* предоставляет разработчику программные интерфейсы для языков *Java*, *Scala*, *Python*, *R*. *Spark* состоит из ядра и нескольких расширений, таких как ***Spark SQL*** (позволяет выполнять *SQL*-запросы над данными), ***Spark Streaming*** (надстройка для обработки потоковых данных), ***Spark MLlib*** (набор библиотек машинного обучения), ***GraphX*** (предназначен для распределённой обработки графов). Система *Spark* может работать как в среде кластера *Hadoop* под управлением *YARN*, так и без компонентов ядра *Hadoop*; она поддерживает несколько распределённых систем хранения — *HDFS*, *OpenStack Swift*, *NoSQL-СУБД Cassandra*, *Amazon S3*.

Приложение *Spark* состоит из одного управляющего процесса, называемого *драйвером* (driver process), и набора исполняющих процессов, называемых *исполнителями* (executor process), которые запускаются на узлах кластера.

В *Spark* вводится концепция ***RDD*** (устойчивый распределенный набор данных) – неизменяемой отказоустойчивой распределенной коллекции объектов, которые могут обрабатываться параллельно. *RDD* представляет собой объект *Scala* и может содержать в себе объекты любых типов. *RDD*

создается посредством загрузки внешнего набора данных из хранилища или распределения коллекции из основной программы.

Над *RDD* могут выполняться операции двух типов:

— ***Трансформации*** — это операции, результатом которых становится преобразование старого RDD в новый, содержащий результат трансформации;

— ***Действия*** — это операции, возвращающие в программу-драйвер значения, полученные в результате вычислений над RDD.

Spark поддерживает концепцию «ленивых вычислений», согласно которой результат вычисляется не сразу после трансформации, а в тот момент, когда он возвращается процессу-драйверу. Подобный подход позволяет в действительности вычислять только те результаты, которые необходимы в данный момент. Вызов действия в приложении *Spark* инициирует запуск соответствующего *задания*. Чтобы определить, что собой представляет это задание, *Spark* анализирует граф *RDDs*, принимающих участие в этом действии, и формирует план выполнения. Этот план начинается с наиболее старых *RDDs* (не зависящих от других *RDDs* или ссылающихся на уже кэшированные данные) и заканчивается на последнем *RDD*, необходимом для получения результатов действия.

2.2.2 Порядок выполнения лабораторной работы

1. Установите соединение с кластером и откройте удаленный рабочий стол. Номер учетной записи выберите в соответствии с номером рабочего компьютера в аудитории. Логин и пароль к учетной записи получите у преподавателя.

2. Создайте в домашней директории папку, в которой будут располагаться Ваши файлы. В имя директории желательно включить фамилии пользователей. В этой директории создайте еще одну папку, в которой будет храниться Spark-проект.

3. Скопируйте файл с исходным кодом *WordCount.java* и набор текстовых файлов в созданную директорию *SparkWordCount* с помощью программы *WinSCP*.

4. Создайте аналогичную папку в системе HDFS:

$ **hadoop fs -mkdir -p ~/ivanov_directory/SparkWordCount**

5. Создайте в системе HDFS директорию, необходимую для хранения исследуемых текстовых файлов:

$ **hadoop fs -mkdir ivanov_directory/SparkWordCount/input**

6. Скопируйте полученные у преподавателя текстовые файлы в папку *input*. Проверьте их наличие после копирования:

$ **hadoop fs -copyFromLocal ~/ivanov_directory/SparkWordCount/*.txt /user/user[0-9]/ivanov_directory/SparkWordCount/input**

$ **hadoop fs -ls /user/user[0-9]/ivanov_directory/SparkWordCount/input**

7. Установите необходимые для работы программы переменные окружения:

$ **export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera/**

$ **export HADOOP_CLASSPATH=$(echo /cloudera/parcels/CDH/jars/*.jar | tr ' ' ':'):.**

8. Перейдите в директорию *SparkWordCount*. Скомпилируйте файл с исходным кодом *WordCount.java*, который представляет программу,

определяющую количество повторений слов в наборе текстовых файлов. Для этого в командной строке выполните следующую команду:

$ **$JAVA_HOME/bin/javac -classpath $HADOOP_CLASSPATH -d . WordCount.java**

9. Соберите Java-пакет из всех полученных в результате компиляции файлов:

$ **$JAVA_HOME/bin/jar cf wc.jar WordCount*.class**

10. Запустите программу для подсчета количества слов в Spark:

$ **spark-submit --class WordCount --master yarn --deploy-mode cluster --executor-memory 512m wc.jar ivanov_directory/ SparkWordCount/input ivanov_directory/SparkWordCount/output**

11. Дождитесь окончания работы задачи. Обратите внимание на ее статус завершения. Представьте скрин вывода. Определите id-задачи в системе yarn (строка вида ***application_1454960730847_0050***). Получите лог работы задачи и результаты ее работы и перенаправьте вывод в локальный файл следующей командой:

$ **yarn logs -applicationId application_1454960730847_0050 > ~/ivanov_directory/SparkWordCount/SparkWordCount.log**

12. По окончанию работы программы проверьте статус ее завершения. Если выполнение программы произошло с ошибками, исправьте их и запустите расчет вновь.

13. Добейтесь успешного завершения программы. Зафиксируйте и проанализируйте консольный вывод приложения. Просмотрите информацию о Spark, предоставляемую Cloudera Manager. На вкладке «History server Web UI» найдите изображение графа DAG,

соответствующего выполнившемуся заданию (см. рисунок 2), и поместите его в протокол.

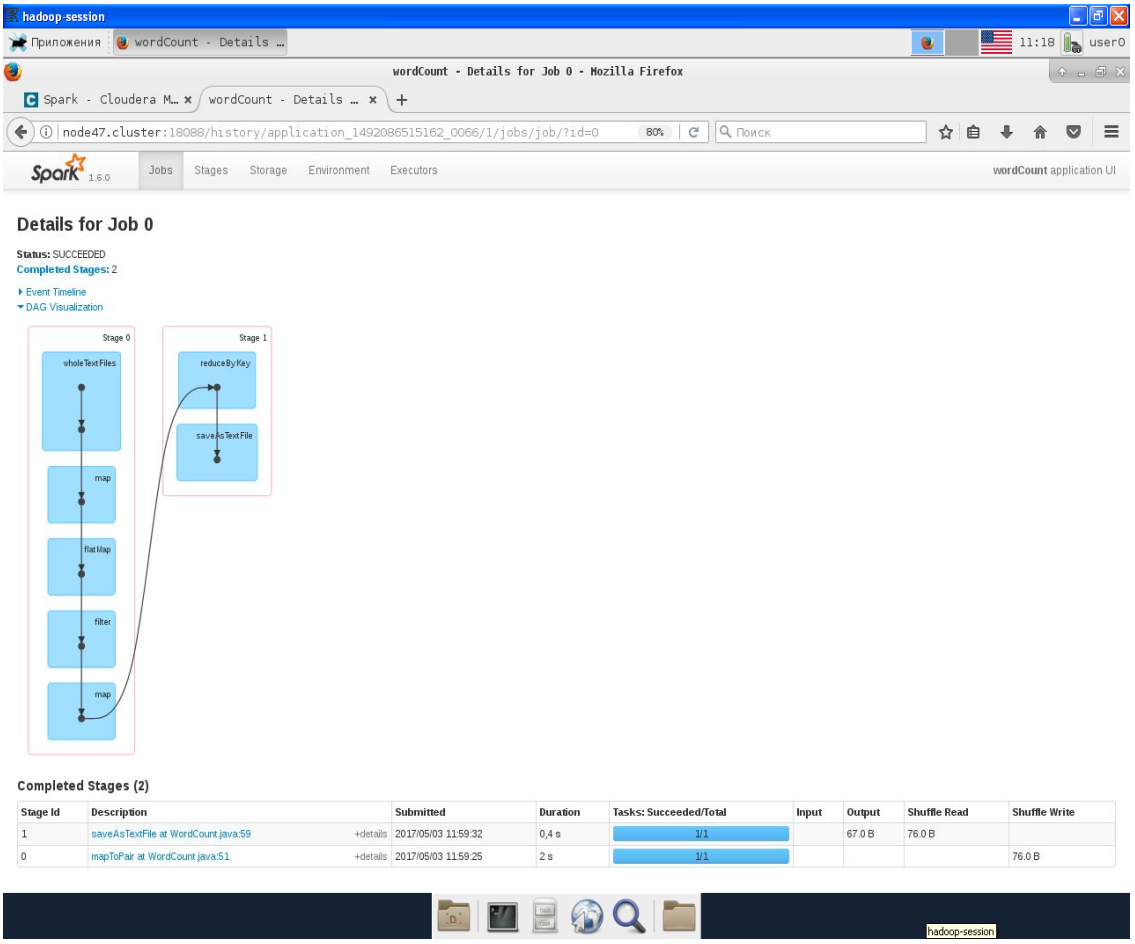

Рисунок 2. Иллюстрация графа DAG в системе Spark

14. Проверьте содержимое директории *output*. Выведите содержимое всех файлов в директории *output* в файл *result.txt* в текущей директории.

$ **hadoop fs -getmerge ivanov_directory/SparkWordCount/output/ result.txt**

15. Просмотрите содержание полученного файла с помощью текстового редактора и проанализируйте его содержимое. Фрагмент файла поместите в протокол.

16. Доработайте программу по подсчету числа вхождения слов в текстовые файлы, добавив в нее требуемое количество MapReduce-фаз, определяющих *количество слов с заданной характеристикой встречаемости в текстах* (см. таблицу 1).

17. Проанализируйте результаты, выводимые модифицированной программой. Доработанный исходный код программы, результаты ее работы, скрины информационных окон Cloudera Manager поместите в протокол. Сделайте выводы.

2.2.3 Домашнее задание

Разработайте программу для решения задачи с использованием технологии MapReduce на базе фреймворка Apache Spark. **Индивидуальные задания по вариантам те же самые, что и для лабораторной работы № 1.** Выполните для разработанной программы пункты 4–18 и занесите результаты в протокол. Сделайте выводы.

2.2.4 Теоретические вопросы к отчету лабораторной работы

1. Фреймворк обработки больших данных Spark, его назначение, функции и отличия от Hadoop MapReduce.

2. Понятие устойчивого распределенного набора данных (RDD). Понятие раздела RDD (partition). Способы создания RDD. Трансформации (transformations) и действия (actions). Кэширование (cache) данных в Spark.

3. RDD и PairRDD: понятие, назначение. Основные трансформации (transformations) и действия (actions) над ними. Понятие кортежа (tuple2), связь между JavaPairRDD и JavaRDD посредством кортежа.

4. Реализация концепции MapReduce в фреймворке Spark. Функции map, flatMap, mapValues, reduce, reduceByKey.

5. Архитектура среды Spark времени выполнения. Модели запуска Spark-приложений (YARN, Mesos, Standalone). Понятие драйвера (driver) и исполнителей (executors). Понятия задания (job), этапа (stage) и задачи (task). Модель ленивых вычислений (lazy) и ее применение в Spark. Понятие ориентированного ациклического графа (Directed Acyclic Graph).

6. Клиентский (client) и кластерный (cluster) режимы работы Apache Spark.

7. Работа с переменными в Spark, понятие замыкания (closure). Аккумуляторы (accumulators), широковещательные (broadcast) переменные.

8. Операция перемешивания данных (shuffle): причины возникновения, влияние на производительность. Класс Partitioner. Пути повышения производительности.

2.3 Лабораторная работа № 3. Изучение технологии NoSQL на основе нереляционной базы данных Apache HBase

Цели работы:

1. Познакомиться с концепцией NoSQL, БД Apache HBase, ее особенностями и областями применения;

2. Изучить архитектуру системы HBase;

3. Получить представление об основных операциях над таблицами и данными в HBase и инструментами взаимодействия с базой данных;

4. Получить навыки создания простых программ, взаимодействующих в Apache HBase с использованием Java API;

5. Получить навыки создания программного обеспечения, обеспечивающего взаимодействие Apache HBase и Apache Spark.

2.3.1 Теоретические основы

Быстрый рост объема накапливаемой информации и ее многообразие привели к проблемам ее обработки с использованием традиционных реляционных баз данных. Строго определенная структура реляционных БД, ранее дававшая преимущества в производительности обработки и гибкости построения запросов к данным, с увеличением объема хранимых данных привела к снижению производительности. Кроме того, не все данные имели табличную структуру, что затрудняло эффективное применение реляционных баз. Также, реляционные БД с трудом масштабируются, что не позволяет в полной мере задействовать все вычислительные ресурсы распределенных систем.

К основным особенностям NoSQL баз данных относят:

— слабые требования к схеме БД и структуре хранимых данных.

— поддержка огромных объемов хранимых данных.

— очень хорошие возможности масштабирования.

Основные типы NoSQL баз данных включают:

— Хранилище «ключ-значение».

— Колоночно-ориентированная БД.

— Документо-ориентированная БД.

— Графовая БД.

HBase — нереляционная распределенная база данных, являющаяся аналогом Google BIgTable. HBase работает поверх распределенной файловой системы HDFS и обеспечивает BigTable-подобные возможности для Hadoop. В HBase данные хранятся 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 байт. Ключи строк таблицы тоже являются байтовыми массивами. Строки таблицы сортируются по ключу строк (первичному ключу таблицы). Сортировка осуществляется в порядке следования байтов. Все обращения к таблице выполняются по первичному ключу.

Столбцы в HBase объединяются в семейства столбцов. Семейства столбцов таблицы должны быть заданы заранее как часть определения схемы таблицы, однако новые члены семейств могут добавляться по мере надобности. Физически все члены семейств столбцов хранятся вместе в файловой системе. Настройки и спецификации задаются на уровне семейств столбцов, поэтому желательно, чтобы все члены семейств имели сходные схемы доступа и характеристики.

2.3.2 Порядок выполнения лабораторной работы

1. Установите соединение с кластером и откройте удаленный рабочий стол. Номер учетной записи выберите в соответствии с номером рабочего компьютера в аудитории.

2. Создайте в домашней директории папку, в которой будут располагаться Ваши файлы. В имя директории желательно включить фамилии пользователей. В этой директории создайте еще одну папку, в которой будет храниться HBase-проект.

3. Скопируйте файлы с исходными кодами *HBase_writer.java* и *JavaWordCount.java*, а также текстовый файл *textfile.txt* в созданную директорию с помощью программы WinSCP.

4. Запустите встроенную консоль системы HBase:

$ **hbase shell**

5. Получите служебную информацию о HBase с помощью команд *status*, *version*, *whoami*. Проанализируйте полученную информацию.

6. С помощью команд, приведенных ниже, введите в HBase таблицу об успеваемости иностранных студентов, представленную в таблице 2, и узнайте ее характеристики. Дайте таблице уникальное имя. Дисциплины должны относиться к **различным семействам столбцов**.

hbase(main):xxx:0> create 'ivanov_table', {NAME => 'Informatics', VERSIONS => 5}, {NAME => 'Mathematics', VERSIONS => 2}

hbase(main):xxx:0> describe 'ivanov_table'

hbase(main):xxx:0> put 'ivanov_table', 'Smith', 'Informatics:Skip', '2'

...

7. Данные команды позволяют создать таблицу HBase с двумя семействами колонок. Первое семейство позволяет хранить пять версий ячейки с различными временными метками, а второе — две версии. Обратите внимание, что колонки в HBase адресуются парой семейство_столбцов:имя_столбца.

8. Просмотрите данные, введенные в таблицу:

hbase(main):xxx:0> get 'ivanov_table', 'Smith'

hbase(main):xxx:0> scan 'ivanov _table'

Таблица 2. Ведомость успеваемости студентов по дисциплинам

Name	Informatics		Mathematics	
	Mark	Skip	Mark	Skip
Smith	98	2	86	4
Williams	65	5	75	2
Brown	84	6	58	6
Davis	80	1	90	0

9. Зафиксируйте полученный результат и удалите таблицу:

hbase(main):xxx:0> disable 'ivanov_table'

hbase(main):xxx:0> drop 'ivanov_table'

10. Выйдите из консоли HBase с помощью сочетания клавиш Ctrl-D. Задайте необходимые для работы переменные окружения:

$ export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera/

$ export HADOOP_CLASSPATH=$(echo /cloudera/parcels/CDH/ jars/*.jar | tr ' ' ':'):.

11. **Допишите** в исходный код программы-клиента, осуществляющей запись информации в HBase с помощью Java API, строки, помещающие в таблицу БД все записи из таблицы 2. Скомпилируйте приложение:

$ $JAVA_HOME/bin/javac -classpath $HADOOP_CLASSPATH HBase_writer.java

12. Запустите программу на исполнение, указав в качестве параметра имя создаваемой таблицы. Найдите среди вывода программы сообщения об успешном выполнении работы (или сообщения об ошибках, которые необходимо исправить).

$ $JAVA_HOME/bin/java -classpath $HADOOP_CLASSPATH HBase_writer <table_name>

13. Вновь запустите консоль HBase и выведите созданную в предыдущем пункте таблицу. Просмотрите ее схему командой *describe*.

14. Проанализируйте программный код приложения, осуществляющего подсчет одинаковых слов в тексте с помощью фреймворка Spark и выводящего результаты работы в HBase (исходный код приложения представлен в файле *JavaWordCount.java*).

15. Создайте в HBase Shell новую таблицу, в которую будут выводиться результаты программы:

hbase(main):xxx:0> create 'ivanov_spark_table', 'Parameters'

16. Скомпилируйте и запустите на исполнение в системе **Apache Spark** программу. Добейтесь ее успешного исполнения. После корректного завершения приложения вновь запустите консоль HBase и проверьте наличие таблицы с результатами.

17. **Создайте** клиентское приложение, соединяющееся с HBase по Java API и получающее информацию о словах в таблице. Добавьте в Ваше приложение возможности вывода содержимого таблицы на экран.

2.3.3 Домашнее задание

1. Спроектируйте и создайте в системе Apache HBase базу данных согласно варианту, самостоятельно выбрав наиболее походящую схему таблицы. Заполните таблицу правдоподобными данными.

2. Разработайте программу, позволяющую считывать данные из таблицы HBase и на их основе выполнять расчет (согласно варианту) с использованием технологии MapReduce на базе фреймворка Apache Spark.

Индивидуальные задания по вариантам

№ варианта	Постановка задач для создаваемой БД и обрабатывающей ее процедуры
1	**База данных учеников общеобразовательной школы.** БД может хранить: класс, в котором обучается ученик, домашний адрес ученика, телефон его родителей, сведения о поведении ученика, данные о победах на олимпиадах и среднюю оценку. Требуется получить число учеников со средней оценкой, большей 4,5 и хорошим поведением.
2	**База данных сотрудников завода.**

	БД может хранить: образование сотрудника, его должность, ФИО его начальника, величину зарплаты, число подчиненных и дату повышения квалификации. Требуется получить число начальников, у которых с даты повышения квалификации прошло больше двух лет.
3	**База данных метеорологических датчиков.** БД может хранить: максимально и минимально возможную температуру, регистрируемую датчиком, максимально и минимально возможную влажность, регистрируемую датчиком, номер паспорта и название завода-изготовителя. Требуется получить название завода-изготовителя датчика, имеющего максимально возможный диапазон рабочих температур.
4	**База данных товаров в магазине.** БД может хранить: число экземпляров товара на складе, число экземпляров товара в упаковке, цена одного экземпляра товара, название фирмы-производителя, срок годности и номер ГОСТа, которому соответствует товар. Требуется получить самый дешевый ГОСТовский товар.
5	**База данных квартир жилого дома.** БД может хранить: этаж, на котором расположена квартира, ФИО владельца, число комнат, площадь квартиры, площадь балкона, материал стен. Требуется получить ФИО владельца трехкомнатной квартиры наибольшей площади.
6	**База данных растений ботанического сада.** БД может хранить: вид растения (деревья, цветы, кустарники…), характеристику его цветов, характеристику его плодов, климатический пояс, требуемый объем полива и площадь территории, занимаемой одним растением. Требуется получить число пустынных растений, требующих менее 1 литра воды в месяц.
7	**База данных транспортных средств.** БД может хранить: класс транспортного средства (легковые машины, грузовики, велосипеды…), его максимальную скорость, грузоподъемность, мощность, максимальное количество перевозимых человек, потребление топлива на 100 км. Требуется получить число автобусов, способных переводить более 20 человек.
8	**База данных клиентов отдела продаж.** БД может хранить: адрес клиента, его телефон, e-mail, покупаемый им товар, количество закупаемого товара, скидка на товар для данного клиента. Требуется получить адрес клиента, купившего более 20 ед. товара и имеющего наибольшую скидку.
9	**База данных ресторанных блюд.** БД может хранить: тип блюда (супы, салаты, напитки…), цену одной порции, массу одной порции, калорийность одной порции, состав блюда и ФИО повара. Требуется получить калорийность самого дешевого супа.
10	**База данных потребителей коммунальных услуг.**

	БД может хранить: адрес потребителя, потребленное им количество электроэнергии, воды, газа, стоимость центрального отопления и полную сумму задолженности. Требуется получить число клиентов, не имеющих центрального отопления, с суммой задолженности более 1000 руб.
11	**База данных книжной продукции издательства.** БД может хранить: название книги, авторов книги, число томов, стоимость издания одного экземпляра, формат книги, ISBN. Требуется получить название книги формата A4 с наибольшим числом авторов.
12	**База данных салона проката автомобилей.** БД может хранить: ФИО клиента, паспортные данные клиента, телефон клиента, номер взятого напрокат автомобиля, марку автомобиля, цену проката. Требуется получить паспортные данные клиента, взявшего напрокат самый дорогой автомобиль Audi.

2.3.4 Теоретические вопросы к отчету лабораторной работы

1. Понятие распределенной системы, проблемы построения распределенных систем. Частичные сбои. CAP-теорема и BASE-принцип. Согласованность строгая (strict) и в конечном счете (eventual).

2. Понятие NoSQL баз данных. Особенности NoSQL БД и отличия их от реляционных БД. Типы NoSQL баз данных. Их особенности, преимущества и недостатки. Примеры.

3. Система HBase, ее особенности. Ситуации, в которых предпочтительно использование HBase.

4. Модель данных HBase. Понятия строк, колонок, семейства колонок. Свойства семейств колонок. Временные метки.

5. Архитектура HBase. MasterServer и RegionServer, их функции. Понятие MemStore и Write Ahead Log, их применение. Понятие региона, правила их образования. Принципы образования новых регионов при добавлении данных в БД. Сервис ZooKeeper. Шардинг, виды шардинга.

6. Хранение данных в HBase. Файлы HFile, их организация. Процедура удаления записей из БД. Minor и major compactions. Основные

операции с данными в HBase.

7. Консольный интерфейс к HBase (HBase Shell) и Java API, их основные функции.

8. Организация обмена данными между Spark и HBase. Метод saveAsNewAPIHadoopDataset. Понятие Bulk loading.

ЗАКЛЮЧЕНИЕ

Если говорить о терминологии, то «Big Data» подразумевает не только данные как таковые, но и принципы обработки больших данных, возможность дальнейшего их использования, порядок обнаружения конкретного информационного блока в больших массивах. Вопросы, связанные с такими процессами, не теряют своей актуальности. Их решение носит важный характер для тех систем, которые многие годы генерировали и копили различную информацию.

Использованные и рекомендуемые источники

1. Лэм, Ч. Hadoop в действии [Текст] / Ч. Лэм. — ДМК Пресс, 2012. — 424с.
2. Apache Hadoop [Электронный ресурс] : офиц. сайт. — Режим доступа : http://hadoop.apache.org/, свободный (дата обращ. 04.05.2017).
3. Джонс, Т.М. Использование Hadoop YARN [Электронный ресурс] / Т.М. Джонс // IBM DeveloperWorks Россия, 2013. – Режим доступа : https://www.ibm.com/ developerworks/ru/library/bd-hadoopyarn/, свободный (дата обращ. 04.05.2017).
4. Кава, А. Введение в YARN [Электронный ресурс] / А. Кава // IBM DeveloperWorks Россия, 2014. – Режим доступа : https://www.ibm.com/developerworks/ ru/library/bd-yarn-intro/ (дата обращ. 04.05.2017).
5. Изучаем Spark: молниеносный анализ данных [Текст] / Х. Карау, Э. Конвински, П. Венделл, М.М. Захария // ДМК Пресс, 2015. — 304 с.: ил.
6. Apache Spark [Электронный ресурс] : офиц. сайт. — Режим доступа : http://spark.apache.org/, свободный (дата обращ. 04.05.2017).
7. Оптимизация заданий Apache Spark. Часть 1 [Электронный ресурс]. – [2015]. – Режим доступа :

http://datareview.info/article/optimizatsiya-zadaniy-apache-spark-chast-1/ (дата обращ. 12.03.2017).

8. Оптимизация заданий Apache Spark. Часть 2 [Электронный ресурс]. – [2015]. – Режим доступа : http://datareview.info/article/optimizatsiya-zadaniy-apache-spark-chast-2/ (дата обращ. 12.03.2017).
9. Уайт, Т. Hadoop: Подробное руководство [Текст] / Т. Уайт. — 3-е изд. — СПб. : Питер, 2013. — 672 с. : ил.
10. Lars, G. HBase. The Definitive Guide [Текст] / G.. Lars — O'Reilly Media, Inc, 2010. — 553 с. : ил.
11. Dimiduk, N. HBase in Action [Текст] / N. Dimiduk, A. Khurana. — Manning Publications Co, 2013. — 335 с. : ил.
12. Apache HBase [Электронный ресурс] : офиц. сайт. — Режим доступа : https://hbase.apache.org, свободный (дата обращ. 04.05.2017).
13. HBase Tutorial [Электронный ресурс] : пособие по HBase. – Режим доступа : https://www.tutorialspoint.com/hbase/ (дата обращ. 04.05.2017).

Учебное издание

Павел Дмитриевич Кравченя
Андрей Евгеньевич Андреев

СИСТЕМЫ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Учебное пособие

Волгоградский государственный технический университет.
400005, г. Волгоград, просп. В. И. Ленина, 28, корп. 1.