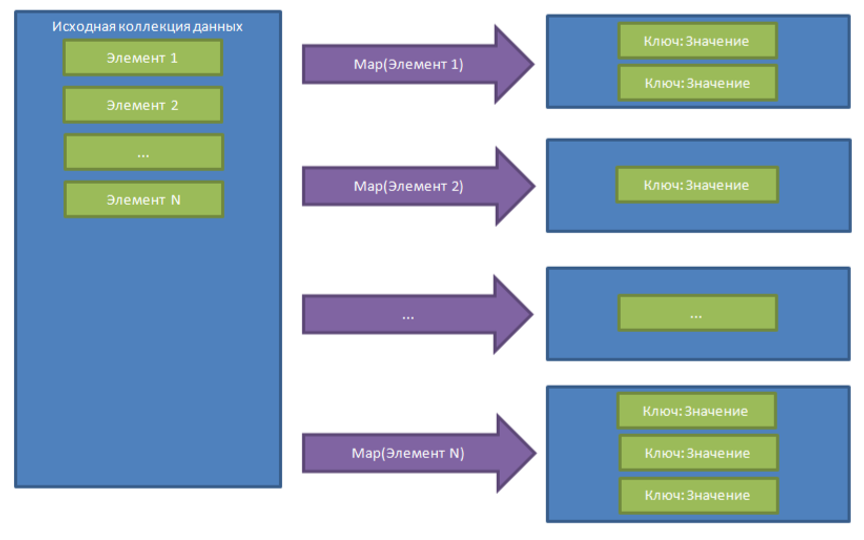
**Лабораторная работа №6**

**РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ. MAPREDUCE В MONGODB**

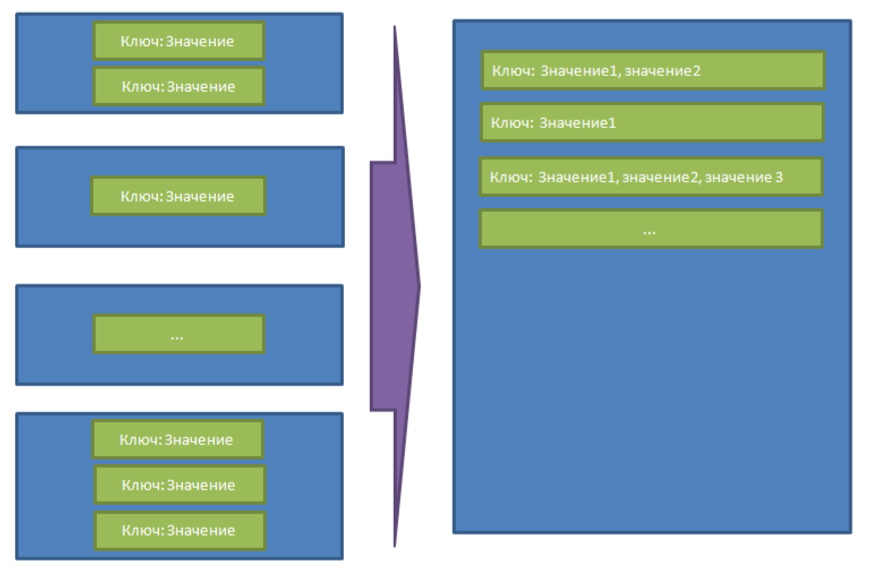
Цель лабораторной работы: изучить модель распределенных вычислений MapReduce. Задачи лабораторной работы: научиться выполнять распределенные операции над документами.

**Теоретическое обоснование**

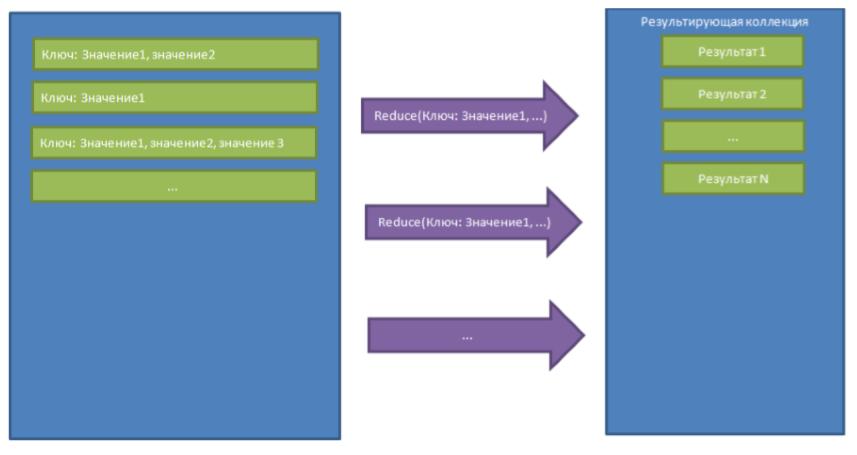
MapReduce – это модель распределённых вычислений, параллельной обработки очень больших объемов данных (измеряемыми петабайтами), в компьютерных кластерах. Компьютеры, входящие в вычислительный кластер называются «узлами». Существуем два типа узлов: главный узел и рабочий. Работа MapReduce состоит из двух шагов: Map и Reduce (отображение и свертка). Map-шаг производит предварительную обработку входных данных. Для этого главный узел получает входные данные задачи, разделяет их на части и передает рабочим узлам для предварительной обработки. На этом шаге входные данные преобразуются в пары ключ/значение (и ключ, и значение могут быть составными). Для этого функция Map применяется к каждому элементу исходной коллекции. Графически работа функции показана на рисунке



После первого шага, алгоритм сортирует все пары ключ/значение, и группирует все значения по ключу. Данный шаг продемонстрирован на рисунке ниже



На втором шаге (Reduce-шаг) происходит свертка предварительно обработанных данных. При свертке на входе получаются ключ и массив значений, порожденный для этого ключа, а на выходе – финальный результат. Функция Reduce выполнится для каждого сгруппированного экземпляра пары ключ/значение. Графически результат работы Reduce-шага представлен на рисунке



После завершения Reduce-шага главный узел получает ответы от рабочих узлов и на их основе формирует результат – решение задачи. И Map и Reduce шаги выполняются распределено на компьютерах кластера. Все операции независимы друг от друга и могут производиться параллельно. Основным преимуществом MapReduce по сравнению с традиционными решениями по обработке данных является производительность. Производительность достигается за счет распараллеливания вычислений и их независимости друг от друга. Кроме того, функции Map и Reduce описываются на языке программирования высокого уровня, что, несомненно, предоставляет широкие возможности по обработке данных.

**Работа с MapReduce в MongoDB**

Для обработки коллекций данных с помощью MapReduce необходимо в консоли mongo выполнить команду «mapReduce» применительно к обрабатываемой коллекции. Синтаксис команды:

db.<коллекция>.mapReduce(<map>, <reduce>,

{

<out> (коллекция),

<query> (Документ),

<sort> (Документ),

<Limit> (Число),

<finalize> (Функция),

<scope> (Документ),

<jsMode> (Логическое),

<verbose> (Логическое)

},

где:

<map> – функция Map написанная на JavaScript;

<reduce> – функция Reduce написана на JavaScript;

<out> – указывает, куда выводить результат выполнения MapReduce;

<query> – определяет критерии отбора с использованием операторов запроса для входных документов Map функции;

<sort> – сортирует входные документы;

<limit> – задает максимальное число документов для возврата из коллекции; <finalize> – JavaScript функция, которая вызывается после исполнения Reduce-шага;

<scope> – определяет глобальные переменные, которые доступны в функциях Map, Reduce и Finalize;

<jsMode> – указывает, требуется ли преобразовывать промежуточные данные в BSON формат;

<verbose> – указывает, следует ли включать информацию о времени выполнения в результирующую информацию.

Первые три параметра функции являются обязательными, остальные – опциональные. При передаче в качестве параметра out имени коллекции все данные из неё будут удалены. Для того чтобы добавить данные в существующую коллекцию необходимо использовать опцию «merge»: {out: {merge: <коллекция>}}

**Порядок выполнения лабораторной работы**

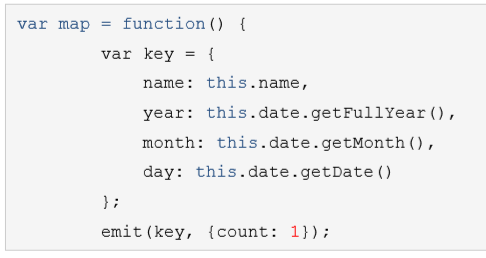
1. Создайте коллекцию для хранения исходных документов для MapReduce.

**> db.selling.find()**

2. Наполните коллекцию данными.

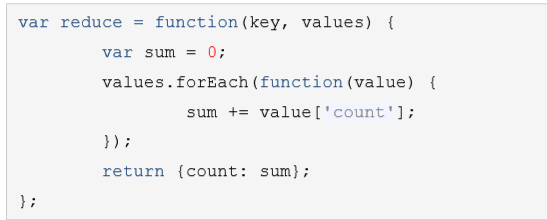
****

3. Создайте map-функцию в консоли (консоль позволяет вводить многострочные конструкции).



}

4. Создайте Reduce функцию в консоли.

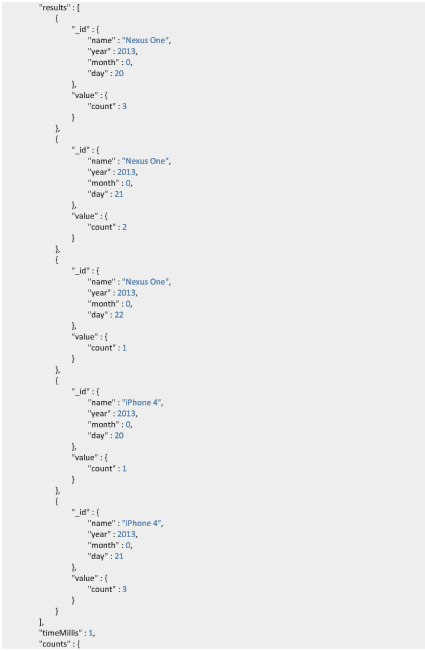


5. Выполните команду mapReduce над коллекцией «selling». Укажите параметр «inline: 1» для вывода результата в консоль.

**>db.selling.mapReduce(map, reduce, {out: {inline: 1}})**

6. Ознакомьтесь с результатом выполнения mapReduce.

> db.selling.mapReduce(map, reduce, {out: {inline: 1}})





7. Выполните команду mapReduce над коллекцией «selling». Для вывода результата в коллекцию параметр «out» принимает имя результирующей коллекции.

**>db.selling.mapReduce(map, reduce, {out: “sellingResult”})**

8. Удостоверьтесь в том, что выполнение функции произошло без ошибок и результат соответствует ожидаемому. Запросите список документов коллекции «sellingResult»:

Сравните список полученных документов с ожидаемым результатом:



Рассмотрим следующий пример и создадим новую коллекцию Writer

**> db.createCollection("Writer")**

Довавим в коллекцию Writer пять документов:

{"Автор": "Пелевин В. О.", "Название": ["Чапаев и Пустота"], "Тип": "Роман"}

{"Автор": "Пелевин В. О.", "Название": ["Бетман Аполло"], "Тип":"Фантастика"}

{"Автор": "Лукьяненко C. В.", "Название": ["Ночной дозор", "Дневной дозор"], "Тип":"Фантастика"}

{"Автор": "Кант И.", "Название": ["Критика чистого разума", "Критика практического разума"], "Тип":"Философия"}

{"Автор": "Пелевин В. О.", "Название": "Generation «П»", "Тип": "Роман"}

Используем MapReduce для вычисления количества произведений каждого типа. Для этого выбираем ключом атрибут "Тип" и создаем функцию map, которая для каждого входного документа генерирует пару (key, value), где ключ (key) получает значение атрибута «Тип», а value — количество произведений — элементов в массиве "Название". При создании map-функции необходимо учесть, что в последнем документе значение атрибута "Название": "Generation «П» " не является массивом и попытка определить его длину приводит к ошибке. Поэтому в функции предусмотрена проверка типа атрибута: typeof (this.Название) == 'object'. Функция map имеет вид:

**> var mapFunc1 = function () {**

**... var key = this.Тип;**

**... if (typeof (this.Название) == 'object')**

**... value= this.Название.length;**

**... else value = 1;**

**... emit (key, value);}**

Задачей функции reduce является суммирование элементов (количество произведений) в массиве значений для каждого значения ключа (тип произведения):

**> var reduceFunc1 = function (keyType, valuesNumb) {**

**... return Array.sum (valuesNumb);}**

Последовательное выполнение обеих функций реализует метод обрабатываемой коллекции db.Writer.mapReduce с сохранением результата в коллекции "TypeNumb":

**> db.Writer.mapReduce ( mapFunc1, reduceFunc1, {out: "TypeNumb"})**

По окончании обработки коллекции Writer метод возвратит статистику решения задачи в формате документа

Запрос db.TypeNumb.find ({}) вернет коллекцию с результатом обработки:

**> db.TypeNumb.find()**

{ "\_id" : "Роман", "value" : 2 }

{ "\_id" : "Фантастика", "value" : 3 }

{ "\_id" : "Философия", "value" : 2 }

В следующем примере рассмотрим обратную задачу подсчета числа авторов для каждой книги в коллекции Writer. Ключом служит название книги, являющееся элементом массива "Название", а значение — число документов (авторов), содержащих в атрибуте-массиве это название.

Дополним коллекцию писателей Writer двумя книгами, имеющими несколько авторов:

({"Автор": "Илья Ильф", "Книги": [{"Название":"Золотой теленок", "Год":1931}, {"Название":"12 стульев", "Год": 1928}]})

({"Автор": "Евгений Петров", "Книги": [{"Название":"Золотой теленок", "Год":1931}, {"Название":"12 стульев", "Год": 1928}]})

Структура новых документов отличается от введенных ранее. Теперь "Название" является атрибутом документа, вложенного в массив "Книги". Наличие атрибута "Книги" позволит выполнить тестирование новых map- и reduce-функций только для этих двух документов.

Создаем функцию для этапа map, которая проверяет в документе наличие атрибута "Книги" и для такого документа генерирует набор пар вида <Ключ, Значение>, где ключом является название книги, а значением 1.

**> var mapFunc2 = function () {if (this.Книги!= null) {//проверка наличия атрибута "Книги"**

**... for (var i = 0; i < this.Книги.length; i++) { emit (this.Книги [i].Название, 1)} } };**

Функция Reduce суммирует значения 1 для каждого ключа (название книги) и записывает в виде документа {<Ключ>, <Сумма единиц>} в выходную коллекцию:

**> var reduceFunc2 = function (keyНазв, valuesКол) { return Array.sum (valuesКол);}**

Для выполнения функций map и Reduce с сохранением результата в коллекции "Книга\_ЧислоАвторов" вызываем метод MapReduce:

**> db.Writer.mapReduce (mapFunc2, reduceFunc2, {out: "Книга\_ЧислоАвторов"})**

На входе map-функции всего было 7 документов. Проверка this.Книги!= null оставила два последних документа. По ним было эмитировано ("emit":4) четыре пары вида (<Название книги>, 1), которые функцией reduce были объединены в две пары и записаны в коллекцию "Книга\_ЧислоАвторов".

Запрос к полученной коллекции возвращает результат обработки:

**> db.Книга\_ЧислоАвторов.find ({}, {})**

{ "\_id" : "12 стульев", "value" : 2 }

{ "\_id" : "Золотой теленок", "value" : 2 }

**Задание для самостоятельной работы**

1. Создайте коллекцию документов, аналогичную рассмотренным выше, для обработки её с помощью MapReduce.

2. Наполните коллекцию документами.

3. Произведите обработку коллекции с использованием модели распределенных вычислений MapReduce.

**Вопросы для защиты работы**

1. Что означает термин MapReduce?
2. Из каких шагов состоит работа MapReduce?
3. Какими преимуществами обладает MapReduce по сравнению с обычными вычислениями?
4. Опишите работу с MapReduce в MongoDB.

**Содержание отчета**

1.Номер и название лабораторной работы

2. Экранные формы, показывающие порядок выполнения лабораторной

работы, и результаты, полученные в ходе её выполнения.

3.Постановка задачи для самостоятельной работы

4. Экранные формы, показывающие порядок выполнения задания для

Самостоятельной работы с соответствующими пояснениями, и результаты, полученные в ходе её выполнения.

5.Ответы на вопросы для защиты