RWTH AACHEN UNIVERSITY

Implementation of Databases
Assignment #7

Due on February 07, 2017

Prof. Dr. rer. pol. Matthias Jarke

Dr. rer. nat. Christoph Quix

Submitted by:

Sanchit Alekh, Idil Esen Zulfikar

MSc. Software Systems Engineering

Problem 1

MapReduce

Define the following queries using the MapReduce pattern. The input to the map function is one document collection which contains all objects of the database, regardless of their type.

1. Compute the average GPA for all students with an SID less than 1.000.

Solution

Map Function

```
var map1 = function(){
        if(this.sid != null && this.sid < 1000 &&
        ...this.type == "student") {
                emit(null,{gpa : this.gpa, count:1});
        }
};
```

Reduce Function

```
var reduce1 = function (key, value){
         var res = {gpa : 0, count : 0};
          for ( var i=0; i<value.length; i++ ){
            res.gpa += value[i].gpa;
            res.count += value[i].count;
        }
  return res.gpa/res.count ;

};
```

Output of Map-Reduce

```
{ "_id" : null, "value" : 2.9850000000000003 }
```

2. For each course, compute the number of students enrolled in this course, and their average grade.

Solution

Map Function

```
map2 = function(){
        if(this.type == "enroll" && this.cid != null){
                if(this.sid <= 0) // if none of students enrolled
                        emit(this.cid,{count:0,grade:0,avr:0});
                else
                        emit(this.cid,{count:1,grade:this.grade,avr:0})
        }
};
```

Reduce Function

```
reduce2 = function(key,values){
        var res = {count:0,grade:0,avr:0};
        for(var i =0; i < values.length; i++){
                res.grade += values[i].grade;
                res.count += values[i].count;
        }
        if(res.count == 0) // if none of students enrolled
                return res;
        else{
                res.avr = res.grade / res.count;
                return res;
        }
```

Output of Map-Reduce

```
{ "_id" : 111, "value" : { "count" : 0, "grade" : 0, "avr" : 0 } }
{ "_id" : 233, "value" : { "count" : 2, "grade" : 8, "avr" : 4 } }
{ "_id" : 456, "value" : { "count" : 3, "grade" : 5.4, "avr" : 1.8 } }
{
        "_id" : 1009,
        "value" : {
```

```
                "count" : 3,
                "grade" : 8.6,
                "avr" : 2.8666666666666667
        }
}
{
        "_id" : 6000,
        "value" : {
                "count" : 3,
                "grade" : 6.7,
                "avr" : 2.2333333333333334
        }
}
```

3. For each Computer Science course (a course from the department with the name 'CS'), get the course name and the average gpa of the students enrolled in the course.

Solution

NoSQL does not have join operation. Therefore, aggregate function of mongoDB has been used. This function performs left join of two collection. In our question, there is only one collection. In order to have multiple collections, the object types that are going to be joined are stored in the separate collections with the codes below.

Store objects that are department type as a collection

```
db.idb_ex7.aggregate( [
          { $match : { type : "dept" } },
          { $group : { _id : "$did", name: { $push: "$name" } } },
          { $out : "mapred_out_3"}
] );
```

Output of stored department objects

```
{ "_id" : 111, "name" : [ "CHM" ] }
{ "_id" : 567, "name" : [ "MB" ] }
{ "_id" : 234, "name" : [ "CS" ] }
```

Store objects that are enroll type as a collection

```
db.idb_ex7.aggregate( [
          { $match : { type : "enroll" } },
          { $out : "mapred_out_4"}
 ] );
```

Output of stored enroll objects

```
{
        "_id" : ObjectId("5893d4690729923f6183751a"),
        "type" : "enroll",
        "cid" : 456,
        "sid" : 123,
        "grade" : 1.7
}
{
        "_id" : ObjectId("5893d4690729923f6183751b"),
        "type" : "enroll",
        "cid" : 456,
        "sid" : 678,
        "grade" : 1
}
{
        "_id" : ObjectId("5893d4690729923f6183751c"),
        "type" : "enroll",
        "cid" : 456,
        "sid" : 1000,
        "grade" : 2.7
}.....
```

Store objects that are student type as a collection

```
db.idb_ex7.aggregate( [
          { $match : { type : "student" } },
          { $group : { _id : "$sid", gpa: { $push: "$gpa" } } },
          { $out : "mapred_out_5"}
 ] );
```

Output of stored student objects

```
{ "_id" : 1122, "gpa" : [ 3.21 ] }
{ "_id" : 1400, "gpa" : [ 2.45 ] }
{ "_id" : 1000, "gpa" : [ 3 ] }
{ "_id" : 1300, "gpa" : [ 3.4 ] }
{ "_id" : 678, "gpa" : [ 3.33 ] }
{ "_id" : 222, "gpa" : [ 2.3 ] }
{ "_id" : 450, "gpa" : [ 2.86 ] }
{ "_id" : 123, "gpa" : [ 3.45 ] }
```

Then, three left join are performed using aggregate function. First join operation is between enroll objects and student objects. The aim is obtaining information of each student enrolled the particular course. The code and output first join operation:

Left join between enroll objects and students objects on sid

```
db.mapred_out_4.aggregate([
    {
      $lookup:
        {
          from: "mapred_out_5",
          localField: "sid",
          foreignField: "_id",
          as: "student_inf"
        }
   },
   { $out : "mapred_out_6"}
]);
```

Output of join operation

```
{
        "_id" : ObjectId("5893d4690729923f6183751a"),
        "type" : "enroll",
        "cid" : 456,
        "sid" : 123,
        "grade" : 1.7,
        "student_inf" : [
                {
                        "_id" : 123,
                        "gpa" : [3.45]
```

```
                }
        ]
}
{
        "_id" : ObjectId("5893d4690729923f6183751b"),
        "type" : "enroll",
        "cid" : 456,
        "sid" : 678,
        "grade" : 1,
        "student_inf" : [
                {
                        "_id" : 678,
                        "gpa" : [3.33]
                }
        ]
}...
```

Second join operation is between course objects and department objects. For each course, department information is obtained.

Left join between course objects and department objects on did

```
db.idb_ex7.aggregate([
    { $match : { type : "course" } }, // using original collection
    {                          // so use only course objects for join
      $lookup:
        {
          from: "mapred_out_3",
          localField: "did",
          foreignField: "_id",
          as: "dept_inf"
        }
   },
   { $out : "mapred_out"}
]);
```

Output of join operation

```
{
        "_id" : ObjectId("58953151c14dcd2942654133"),
```

```
        "type" : "course",
        "cid" : 456,
        "name" : "Desing Interactive Systems",
        "did" : 234,
        "dept_inf" : [
                {
                        "_id" : 234,
                        "name" : [
                                "CS"
                        ]
                }
        ]
}
{
        "_id" : ObjectId("58953151c14dcd2942654134"),
        "type" : "course",
        "cid" : 1009,
        "name" : "Differential Equations",
        "did" : 111,
        "dept_inf" : [
                {
                        "_id" : 111,
                        "name" : [
                                "CHM"
                        ]
                }
        ]
}...
```

Final join operation is between course objects that have department information for each course and enroll objects that have student information for each enrollment. For each course, enrollment information is placed as an object to the collection.

Left join between course objects and enroll objects on cid

```
db.mapred_out.aggregate([
    {
      $lookup:
        {
          from: "mapred_out_6",
          localField: "cid",
```

```
          foreignField: "cid",
          as: "enroll_inf"
        }
   },
   { $out : "mapred_out"}
]);
```

Output of join operation

```
{
        "_id" : ObjectId("58953151c14dcd2942654133"),
        "type" : "course",
        "cid" : 456,
        "name" : "Desing Interactive Systems",
        "did" : 234,
        "dept_inf" : [
                {
                        "_id" : 234,
                        "name" : [
                                "CS"
                        ]
                }
        ],
        "enroll_inf" : [
                {
                        "_id" : ObjectId("5893d4690729923f6183751a"),
                        "type" : "enroll",
                        "cid" : 456,
                        "sid" : 123,
                        "grade" : 1.7,
                        "student_inf" : [
                                {
                                        "_id" : 123,
                                        "gpa" : [
                                                3.45
                                        ]
                                }
                        ]
                },
                {
                        "_id" : ObjectId("5893d4690729923f6183751b"),
```

```
                        "type" : "enroll",
                        "cid" : 456,
                        "sid" : 678,
                        "grade" : 1,
                        "student_inf" : [
                                {
                                        "_id" : 678,
                                        "gpa" : [
                                                3.33
                                        ]
                                }
                        ]
                },
                {
                        "_id" : ObjectId("5893d4690729923f6183751c"),
                        "type" : "enroll",
                        "cid" : 456,
                        "sid" : 1000,
                        "grade" : 2.7,
                        "student_inf" : [
                                {
                                        "_id" : 1000,
                                        "gpa" : [
                                                3
                                        ]
                                }
                        ]
                }
        ]
}
```

Finally, the collection that is going to use for map-reduce has been obtained. The map-reduce functions below are used to obtain CS department courses and average gpa of students who enrolled this particular CS courses.

```
var map3 = function(){
  if(this.dept_inf != null && this.enroll_inf != null){
    if(this.dept_inf[0].name[0] == "CS"){
    //eliminate courses that their department is not CS
      for(var i =0; i < this.enroll_inf.length; i++){
        if(this.enroll_inf[i].student_inf != null){
```

```
                emit(this.cid,{name:this.name,
                     deptname:this.dept_inf[0].name[0],
                     gpa:this.enroll_inf[i].student_inf[0].gpa[0],
                     count:1});
        }
      }
    }
  }
};
```

Reduce Function

```
var reduce3 = function(key,values){
  res = {name:null,deptname:null,avrgpa:0,count:0,};
  for(var i =0; i < values.length; i++){
    res.avrgpa += values[i].gpa;
    res.count += values[i].count;
  }
  res.avrgpa = res.avrgpa / res.count;
  res.name = values[0].name;
  res.deptname = values[0].deptname;
  return res;
};
```

Output of Map-Reduce

```
{
        "_id" : 456,
        "value" : {
                "name" : "Desing Interactive Systems",
                "deptname" : "CS",
                "avrgpa" : 3.2600000000000002,
                "count" : 3
        }
}
```

Problem 2

PageRank

Consider the following link graph of the websites UMIC (1), DBIS (2), RWTH Aachen University (3), B-IT Research School (4) and FIT Fraunhofer (5):

1. Write down the transition matrix.

Solution

The transition matrix is given as follows:

$$M = \begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Explanation: The above matrix is calculated by following method. For r_{ij}, if there doesn't exist a connection from j to i, then $r_{ij} = 0$, otherwise it is equal to

$$\frac{1}{n(outflows\ from\ j)}$$

2. What is the approximate final page rank vector and which page would be ranked the highest? Hint: you can achieve that by continuously multiplying the transition matrix to the vector until an approximate fixpoint is reached.

Solution

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.09 \\ 0.39 \\ 0.29 \\ 0.19 \\ 0.04 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.09 \\ 0.39 \\ 0.29 \\ 0.19 \\ 0.04 \end{bmatrix} = \begin{bmatrix} 0.1505 \\ 0.3105 \\ 0.2905 \\ 0.1705 \\ 0.0780 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.1505 \\ 0.3105 \\ 0.2905 \\ 0.1705 \\ 0.0780 \end{bmatrix} = \begin{bmatrix} 0.1347 \\ 0.3342 \\ 0.2952 \\ 0.1737 \\ 0.0621 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.1347 \\ 0.3342 \\ 0.2952 \\ 0.1737 \\ 0.0621 \end{bmatrix} = \begin{bmatrix} 0.1407 \\ 0.3259 \\ 0.2949 \\ 0.1717 \\ 0.0668 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.1407 \\ 0.3259 \\ 0.2949 \\ 0.1717 \\ 0.0668 \end{bmatrix} = \begin{bmatrix} 0.1389 \\ 0.3285 \\ 0.2951 \\ 0.1723 \\ 0.0652 \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} 0 & 0.2 & 0.25 & 0 & 0 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0.5 \\ 0.5 & 0.2 & 0.25 & 0.5 & 0 \\ 0 & 0.2 & 0.25 & 0 & 0.5 \\ 0 & 0.2 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.1389 \\ 0.3285 \\ 0.2951 \\ 0.1723 \\ 0.0652 \end{bmatrix} = \begin{bmatrix} 0.1393 \\ 0.3279 \\ 0.2951 \\ 0.1722 \\ 0.0655 \end{bmatrix} \tag{7}$$

At this point, we have reached an approximate fixpoint, as the *PageRank* values no longer change by a substantial amount. Therefore, we can say that the matrix given by *Equation 7* is the final *PageRank Vector*

3. Define a data structure to represent documents (identified by a URL), their outgoing links, and their page rank in a JSON document. Sketch a MapReduce implementation of the Page Rank algorithm using some form of Java/JavaScript pseudo code, similar to the syntax used in the lecture. Preferably, implement the algorithm in MongoDB and test it with a sample collection.

Solution

JSON Data Structure to represent the Webpages

```
{
pageID : 1,
```

```
pageURL : 'umic.rwth-aachen.de'
links : [{pageID: 2, pageURL : 'dbis.rwth-aachen.de', type : in },
         {pageID: 2, pageURL : 'dbis.rwth-aachen.de', type : out }, ..]
trans : [0   0   0   0   0]
pageRank : 0.2
}
```

Map Function

```
map = function(){
    if (this.pageID >= 1){
        emit(this.pageID, {pageURL:this.pageURL,
                           links: this.links,
                           trans: this.trans,
                           pageRank: this.pageRank});
    }
}
```

Reduce Function 1 : calculates the Transition Matrix

```
reduce = function(key, values) {
    output = { links: null , trans: null};
    values.forEach(function(value){
        output.links = value.links;
        output.trans = value.trans;
    if(output.trans != NULL){
    // update Transition Matrix
        for i in 1 to output.trans.length(){
            if(exists values.links.pageID such that pageID = i
               && values.links.type = 'in'){
                numLinks = number of outlinks where pageID = i;
                output.trans[i] = 1/numlinks;
                }
            else{
                output.trans[i] = 0
            }
        }
    }
    return output;
```

```
    )};
}
```

Reduce Function 2 : calculates the PageRank from Transition Matrix

```
reduce = function(key, values) {
    output = {trans: null, pageRank = 0};
    values.forEach(function(value){
        output.trans = value.trans;
        output.pageRank = value.pageRank;
    if(output.trans != NULL){
    // calculate fixpoint for pageRank
        trans[] = {0}
        pageRank[] = {0};
        for j in 1 to output.trans.length(){
            pageRank[j] = retrieve document.pageRank where
                          document.pageID = j;
            trans[j] = retrieve document.trans where
                          document.pageID = j;
        }
        do{
        temp = output.pageRank;

        // Matrix Multiplication (symbolically x)
        updatedPageRank[] = trans[] x  pageRank[];
        // update PageRanks for all documents
        for j in 1 to output.trans.length(){
            update document.pageRank to pageRank[j] where
            document.pageID = j;
        }
        output.pageRank = updatedPageRank[key];
        }
        while(output.pageRank - temp >= |0.001|);
    }
    )};
    return output;
}
```