DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II: Bases de datos NoSql

Bases de Datos
Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Raul Benitti	592/08	raulbenitti@gmail.com
Damian Castro	326/11	ltdicai@gmail.com
Leandro Matayoshi	79/11	leandro.matayoshi@gmail.com
Javier San Miguel	786/10	javiersm00@gmail.com

1. Introducción

En el presente trabajo desarrollaremos un acercamiento práctico a las bases de datos NoSQL. Para tal fin utilizaremos MongoDB, una base de datos de documentos, y revisaremos tres puntos básicos de uso:

- Implementación de un modelo conceptual, guiado por las consultas directas que se desean realizar sobre los datos.
- Implementación de consultas por medio de MapReduce, uno de las tecnologías estandarte de las bases de datos orientadas a documento
- Implementación de escalado horizontal, conocido como sharding.

Como último punto, desarrollaremos brevemente como cambiarían las soluciones implementadas en MongoDB si se utilizara una base de datos de tipo Key-Value.

2. Modelo

Como regla general, el modelo está pensado en función de las consultas que se desea responder. Para ello, utilizamos la desnormalización y la redundancia de datos como mecanismos válidos para poder resolver las consultas rápida y eficientemente.

Esto produce como consecuencia la posibilidad de producir inconsistencias en la base de datos. Por lo tanto, es responsabilidad del programador realizar los chequeos correspondientes a la hora de insertar, modificar o eliminar datos.

2.1. Empleados que atendieron clientes mayores de edad

Para poder satisfacer los requerimientos de la query, mantenemos en la colección *Empleados* una lista de clientes, que contiene los identificadores de cada uno. Para esto utilizamos 2 arreglos, diferenciando entre clientes mayores y menores de edad.

```
{
  "_id" : ObjectId("5622bf41228da935bd5e0a6a"),
  "nroLegajo" : 234,
  "nombre" : "Pepito Suarez",
  "clientes_mayores" : [
    {
      "_id" : ObjectId("5622c1d3228da935bd5e0a6b"),
      "fecha" : ISODate("2015-10-01T00:00:00Z")
    }
  ],
  "clientes_menores" : [ ],
  "sectores" : [
    {
      "sector" : "Comestibles",
      "tarea" : "Gerente"
    },
    {
      "sector" : "Indumentaria deportiva",
      "tarea" : "Supervisor"
    }
  ]
}
```

Listing 1: Ejemplo Empleado

Consulta: db.empleados.find({"clientes_mayores": {$exists: true, $not: {$size: 0}}})

2.2. Artículos más vendidos

Agregamos en un atributo *cant_unidades_vendidas* que nos permite encontrar el máximo muy eficientemente. No mantenemos información acerca de los clientes que compraron ese producto, ya que esta última colección es la encargada de llevar ese registro.

Consulta:

En primer lugar, realizamos una agregación con un único grupo: *_id: null*, mediante la cual encontramos el máximo entre todas las cantidades vendidas. Luego realizamos una nueva query, fitrando por igualdad sobre dicha máxima cantidad obtenida.

```
{
  "_id" : ObjectId("5622d234228da935bd5e0a6f"),
  "codBarras" : 342391,
  "nombre" : "Pro Speed Z-7",
  "sector" : "Calzado",
  "cant_unidades_vendidas" : 0
}
```

Listing 2: Ejemplo Artículo

1. max_cant_unidades_vendidas = (db.articulos.aggregate([{$group : {_id: null, max : {$max : "$cant_unidades_vendidas"}}}])).next().max
2. db.articulos.find({"cant_unidades_vendidas": max_cant_unidades_vendidas})

2.3. Sectores donde trabajan exactamente 3 empleados

La colección *Sectores* mantiene una lista con los ids de los empleados que trabajan en él.

```
{
  "_id" : ObjectId("5622df91228da935bd5e0a75"),
  "codSector" : "Comestibles",
  "empleados" : {
    "lista" : [ ObjectId("5622bf41228da935bd5e0a6a") ]
  }
}
```

Listing 3: Ejemplo Sector

Consulta: db.sectores.find({"empleados.lista": {$size: 3}})

2.4. Empleado que trabaja en más sectores

Consulta:

La consulta es análoga a "artículos más vendidos". En primera instancia encontramos el máximo, y luego seleccionamos aquellos documentos con valor igual a dicho máximo.

1. var max = db.empleados.aggregate([{$group: {_id:null, max: {$max: {$size: "$sectores"}}}}]).next().max
2. db.empleados.find({sectores: {$size: max}})

2.5. Ranking de los clientes con mayor cantidad de compras (total de unidades)

El cliente mantiene una lista con los artículos que compra. Cuando el cliente compra una cantidad determinada de un producto, se agrega una nueva entrada al final de *lista*, sumando al valor de total

la cantidad de unidades compradas. Este campo admite repetidos sobre *id*, lo cual agrega eficiencia a la hora de insertar una nueva compra.

```
{
  "_id" : ObjectId("5622c1d3228da935bd5e0a6b"),
  "dni" : 28012849,
  "nombre" : "Julio Jericho",
  "edad" : 23,
  "articulos" : {
    "total" : 4,
    "lista" : [ {"id": ObjectId("ff20ef41228da935bd5583bd"), "cantidad": 3},
                {"id": ObjectId("4729bce098bbddee98100acc"), "cantidad": 1}
              ]
  }
}
```

Listing 4: Ejemplo Cliente

Consulta:
El ranking está determinado por un ordenamiento descendente sobre el total de compras realizadas por cada cliente.

db.clientes.aggregate([{$sort : {“articulos.total”: -1} }])

2.6. Cantidad de compras realizadas por clientes de la misma edad

Consulta:
Agrupamos los documentos por edad proyectando el total de artículos comprados. Finalmente realizamos una agregación sobre cada grupo, sumando *total*.

db.clientes.aggregate([{$project: {“art_total”: “$articulos.total”, “edad”: 1}}, {$group: {_id: “$edad”, total: { $sum: “$art_total”}}}])

3. MapReduce

3.1. Cantidad de disposiciones de tipo resoluciones realizadas en Abril del 2013

Por cada disposición, si cumple con las restricciones emitimos como clave: “Resoluciones”, y 1 como valor. El reduce consiste simplemente en la suma de todos los values.

```
var m = function(){
  var date = convertDate(this.FechaDisposicion);
  if(date){
    var month = date.getMonth();
    if (month == 3 && this.Tipo == "Resoluciones"){
      emit(this.Tipo, 1);
    }
  }
}

var r = function(key,values) {
    return Array.sum(values)
}

printjson(db.disposiciones.mapReduce(m,r,{out: {inline: 1},
          scope: {convertDate: convertDate}}));
```

3.2. Cantidad de disposiciones por tipo definido

Este caso consiste en un agrupamiento por “Tipoçon la suma de la cantidad de documentos como valor. Por lo tanto, la clave en este caso es el tipo de la disposición. Al igual que en el caso anterior, se emite 1 como valor por cada documento y al reducir se suman todos los valores.

```
var m = function(){
  if (this.Tipo != ""){
    emit(this["Tipo"], 1);
  }
}
var r = function(key, values) {
  return Array.sum(values);
}

printjson(db.disposiciones.mapReduce(m,r,{out: {inline: 1}}));
```

3.3. Fecha más citada para todos los informes

Utilizamos la función convertDate para transformar las distintas fechas a un formato común. Las fechas se presentan como day-month-year, day/month/year o en formato iso: 2015-11-21T01:51:17Z.

Las claves son las fechas normalizadas, y emitimos 1 como valor tanto por la *FechaBOJA* como por la *FechaDisposicion*.

```
var m = function(){
  var fecha_boja = convertDate(this.FechaBOJA);
  var fecha_disp = convertDate(this.FechaDisposicion);
  if (fecha_boja){
    emit(fecha_boja, 1);
  }
  if(fecha_disp){
    emit(fecha_disp, 1);
  }
}
```

```
var r = function(key, values) {
  return Array.sum(values);
}


var res = db.disposiciones.mapReduce(m,r,{out: {inline: 1},
                                    scope: {convertDate: convertDate}})
printjson(res);

//Find max
var max = null;
for(var idx = 0; idx < res.results.length; ++idx){
  var item = res.results[idx];
  if(max){
    if (Math.max(max.value, item.value) != max.value) max = item;
  }
  else{
    max = item;
  }
}

printjson(max);
```

3.4. Devolver la mayor cantidad de páginas utilizadas por cada tipo de disposición

La consulta consiste en un agrupamiento sobre tipo de disposición con agregación sobre el máximo de las páginas utilizadas.

Por lo tanto emitimos el tipo de documento como clave y la cantidad de páginas utilizadas como valor. El reduce encuentra el máximo sobre la lista de valores.

```
var m = function(){
  var pagina_inicial = this["PaginaInicial"],
    pagina_final = this["PaginaFinal"];
  emit(this["Tipo"], pagina_final - pagina_inicial)
}

var r = function(key, values){
  var max = 0;
  // Calculo del maximo
  for(var i = 0; i < values.length; ++i){
    if (values[i] > max){
      max = values[i];
    }
  }
  return max;
}

var res = db.disposiciones.mapReduce(m, r, {out: {inline: 1}})
printjson(res);
```

3.5. Función para convertir las fechas

```
function convertDate(str){
  if(str == "") return null;
  if(/^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}Z?$/.test(str)){
    return new Date(str.replace(/^00/, "20"));
  }
  else if(/^\d{2}[-\/]\d{2}[-\/]\d{4}$/.test(str)){
    var parts = str.split(/[-\/]/);
    var aux = parts[0];
    parts[0] = parts[2];
    parts[2] = aux;
    return new Date(parts.join("-"));
```

```
  }
  else{
    return null;
  }
}
```

4. Sharding

El objetivo de esta sección es analizar la distribución de la carga en una colección repartida entre varios shards.

La estructura de los documentos es la siguiente:

```
{
  "nombre": nombrePersona
  "password": passwordPersona
  "codigo_postal": getRandomIntInclusive(1, 1000000),
  "genero": generoPersona,
  "edad": edadPersona
  "fecha_creacion": fechaCreacion
}
```

Listing 5: Ejemplo persona

La shard key se establece a partir del atributo *codigo_postal*, generado de forma pseudoaleatoria. Los documentos son agregados en bloques de 20.000 documentos, hasta alcanzar un total de 500.000. El tamaño total de la colección es aproximadamente 115 MB. Esto genera un total de 25 mediciones por experimento.

Es posible generar los shards de 2 formas posibles sobre la shard key:

- Por rango
- Por hash

Por cada uno de estos tipos realizamos un experimento con 2 y 3 shards.

4.1. Partición basada en rango

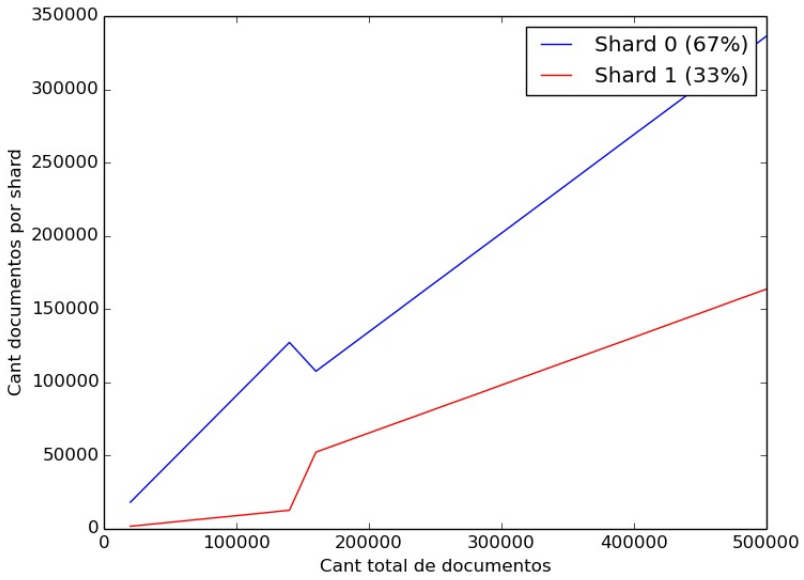

Figura 1: 2 Shards utilizando partición de keys por rango

La carga no se reparte uniformemente. El shard 0 mantiene un aproximadamente un 67 % de los datos a lo largo de todo el experimento, mientras que el segundo shard mantiene el 33 % de los datos.

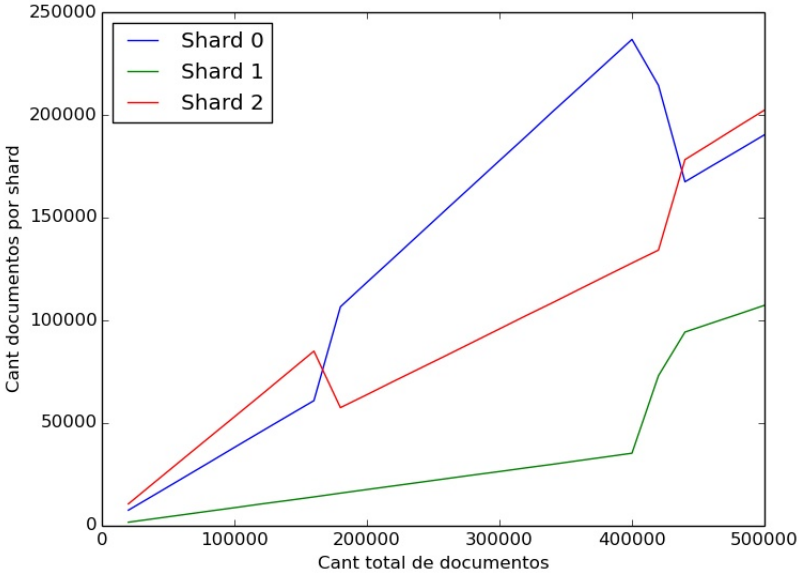

Figura 2: 3 Shards utilizando partición de keys por rango

La existencia de un tercer shard favorece al balance de la carga, aunque sigue habiendo un shard que contiene pocos datos.

4.2. Partición basada en hash

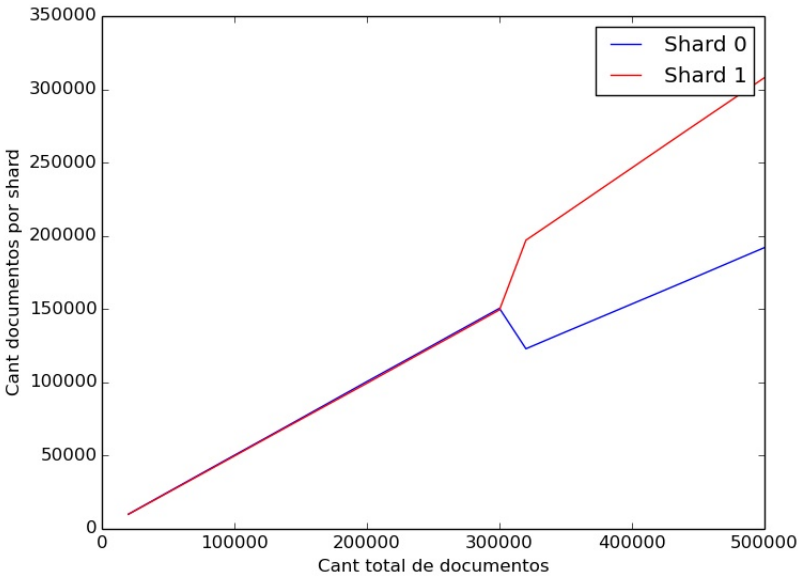

Figura 3: 2 Shards utilizando partición de keys basada en hash

La carga se distribuye de manera uniforme hasta un punto de quiebre, en donde un shard queda con más carga que el otro.

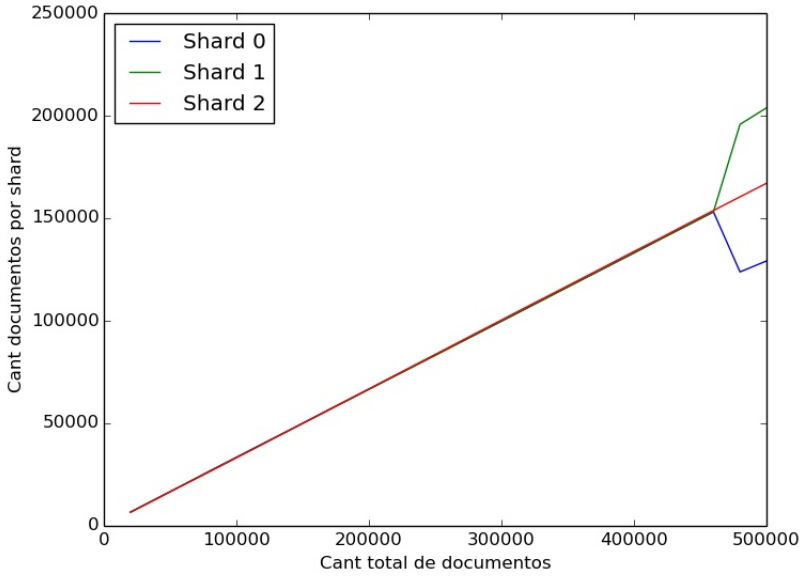

Figura 4: 3 Shards utilizando partición de keys basada en hash

Nuevamente, la existencia de un tercer shard aporta estabilidad al balanceo. En este caso, la distribución se mantiene uniforme casi hasta el final de la experimentación. En dicho punto, los shards quedan con una distribución levemente diferente.

4.3. Conclusión

Para valores de shard keys generados de forma pseudoaleatoria es conveniente tener una clustering key basada en hash para mantener el balanceo uniforme por sobre una con rango. La existencia de mayor cantidad de shards favorecen a la uniformidad de la distribución.

5. Key-Value como alternativa

5.1. General

Elegimos ejemplificar los cambios que implicaría utilizar una base de datos Key-Value. Este tipo de bases está muy relaciono con las bases de datos Document, consideradas como una versión más potente de las primeras. Ambas mantiene una _clave_ relacionada con un único _valor_, y esta relación define el límite de atomicidad. Además, si bien en ambos tipos de bases de datos pueden modelarse relaciones entre entidades, la coherencia y validez de estas relaciones deben ser mantenidad por el programador desde el código de la aplicación.

La diferencia fundamental entre ambos tipos radica en que, en las bases Key-Values, el _valor_ solo se considera como una cadena de bits sin semántica, por lo que cualquier información que allí se guarde resulta inútil a fines de consulta y procesamiento directo: en una base Key-Value, la única manera de acceder a un dato es sabiendo bajo que clave se almacena. En cambio, las bases de datos de documentos sí otorgan una semántica a los valores relacionados a una clave - la idea de un documento con campos definidos. Esto les permite proveer operaciones de búsqueda y agregación más cercanas a aquellas encontradas en bases de datos relacionales.

5.2. Punto 1

Como en las bases Key-Value no hay acceso posible a los datos si no se especifican las claves, a diferencia del modelo realizado en Document DB, necesitamos mantener información explícita sobre las claves de las entidades que vamos a almacenar. Por esto, utilizaríamos un bucket Claves con

claves[“empleados”] → [nroLegajo]
claves[“productos”] → [codigoDeBarra]
claves[“sectores”] → [codigoSector]
claves[“clientes”] → [dni]

Las bases Key-Value tampoco nos proveen métodos para operar sobre los valores almacenados por clave, puesto que la parte del valor es considera como una caja negra. Por esto, si se quiere tener una estructura de clave-valor que no sea extremadamente rigida (como sería el caso de guardar claves-valores del estilo [“empleadosQueAtendieronAdultos”] → {[nroLegajo]}), necesitamos poder iterar sobre las claves y operar con los valores por fuera de la base.

Por ejemplo, podriamos definir la siguiente estructura (incompleta) con los datos necesarios para las consultas requeridas:

1. empleados[< *nroLegajo* >:“clientes”] → {“adultos”: int, “total”: int, “clientesAtendidos”: [dni]}
2. articulos[< *codigoDeBarra* >] → {“cantidadVendida”: int}
3. sectores[< *codigoSector* >] → {“cantidadEmpleados”:int, “empleados”:[nroLegajo]}
4. clientes[< *dni* >] → {cantidadCompras: int}
5. empleados[< *nroLegajo* >:“sectores”] → {“cantidadSectores”: int, “sectores”:[codigoSector]}
6. clientes[“compras”:< *edad* >] → “cantidadCompras”: int

Entonces, para responder la consulta “Los empleados que atendieron clientes mayores de edad”, el programador debería:

- conseguir los nroLegajo de los empleados utilizando claves[“empleados”]
- para cada nroLegajo, obtener el json almacenado en empleados[< *nroLegajo* >:“clientes”]
- mantener una lista de los empleados (nroLegajo) con mayor valor en el campo “adultos”del json obtenido.

5.3. Punto 2

Como comentamos anteriormente, las bases de datos Key-Value no proveen métodos para realizar operaciones sobre la información almacenada en los valores. Por lo tanto, el esquema de Map Reduce debería ser implementado en una capa superior, ya sea directamente por el programador o por una librería. Si no existe una librería que provea tal servicio, podría resultar mas conveniente proceder tal como lo haríamos para las consultas del punto 1.

5.4. Punto 3

El concepto de Sharding puede aplicarse de la misma manera tanto a bases Key-Value como Document, y deben tenerse en cuentas las mismas consideraciones sobre la distribución de las claves elegidas para realizar las particiones. Existe, sin embargo, una diferencia importante a considerar: dependiendo de como se manejen los atributos de una entidad en Key-Value (agregados en una clave→valores o desagregados en varias clave:atributo→valor), pueden darse casos en que una misma entidad tenga sus atributos repartidos entre distintos shards.

6. Conclusiones

En el transcurso del desarrollo de este trabajo nos enfrentamos con problemas de diseño e implementación típicos en el uso de bases NoSQL. Entre ellos:

- Representar la información para prescindir del uso de join, tratando de mantener un balance entre la consistencia de los datos y la atomicidad de las operaciones.
- Extraer información utilizando la técnica de MapReduce, en la cual el propio programador debe cumplir un rol que suele relegarse al motor de bases de dato relacionales
- Dividir de forma adecuada los datos a fin de lograr escalabilidad.

A partir de todo esto, llegamos a entender que las bases de datos NoSQL pueden ser muy útiles en situaciones particulares en las que, entre otras cosas:

- Las consultas que se realizaran sobre los datos son prefijadas o predecibles
- No se requiera realizar consultas arbitrarias
- La consistencia de la informacion no es un factor relevante (aunque existen bases NoSQL que cumplen con ACID, no son la mayoría ni las más utilizadas)
- Se espera que el sistema deba procesar cantidades extremadamente grandes de información de manera eficaz
- Se espera que millones de usuarios utilicen el sistema de manera concurrente

Todas estas características no implican descartar el uso de bases de datos relaciones sino que, si dichas circunstancias están presente, permiten evaluar la conveniencia de dejar de lado la complejidad en que incurren las bases relacionales que cumplen con las propiedades ACID por otras bases con una mecánica simplificada a costa de no hacerlo.

En definitiva, las bases de datos NoSQL no vienen a reemplazar a las bases relacionales - las complementan, poniendo a disposición de los desarrolladores más herramientas con las que encontrar soluciones eficaces a los problemas cada vez más complejos con los que se enfrenta el mundo de la administración de información.