DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Base de datos NOSql

Trabajo Práctico 2

Base de datos

Integrante	LU	Correo electrónico
Guillamon, Gonzalo	97/12	gonzaguillamon@gmail.com
Toffoletti, Luis	827/11	luis.toffoletti@gmail.com
Zanollo, Florencia	934/11	florenciazanollo@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Contents

1 Introducción

2 Consultas SQL

3 Map Reduce

MapReduce es un modelo de programación para el procesamiento y creación de grandes colecciones de datos. Consiste de dos partes:

- Map() se encarga de filtrar y organizar los datos, produciendo una lista de pares (clave, valor) por cada llamada. Luego de esto el framework agrupa los pares con la misma clave, creando un grupo por cada una de las diferentes claves generadas.
- Reduce(clave, valores) aplica funciones sobre los valores agrupados en el map. Por ejemplo, suma todos los valores, saca el promedio, etc. Retornando un resultado por cada clave.

Notar que estos procedimientos (Map y Reduce) son paralelizables, con lo cual pueden ser distribuidos en diferentes unidades (por ejemplo clusters) para su procesamiento.

A continuación listamos todas las consultas a implementar, utilizando MapReduce, requeridas, con su respectivo código y una pequeña explicación del mismo.

3.1 El importe total de ventas por usuario

```
1  var mapFunctionA = function(){
2    emit(this.vendedor, this.importe)
3  };
4
5  var reduceFunctionA = function(userId, importes){
6    return Array.sum(importes);
7  }
8
9  db.Ventas.mapReduce( mapFunctionA, reduceFunctionA, {out: "mapReduceA"} )
10
11 db.mapReduceA.find()
```

linea 9 Se aplica el MapReduce sobre la colección Ventas.

linea 2 Como clave se eligió el id del usuario que realiza la venta (i.e. el e-mail del vendedor) y el valor es el importe de la misma.

linea 5 Con esto se obtiene una lista por vendedor con todos los importes de sus compras.

linea 6 Los cuales son sumados y se retorna dicha suma.

3.2 La reputación historica de cada usuario según la calificación

```
1  var mapFunctionB = function(){
2    emit(this.vendedor, this.calificacionVendedor);
3    emit(this.comprador, this.calificacionComprador);
4  };
5
6  var reduceFunctionB = function(userId, calificaciones){
7    suma = Array.sum(calificaciones);
8    promedio = suma / calificaciones.length
9    return promedio;
10 }
11
12 db.Ventas.mapReduce( mapFunctionB, reduceFunctionB, {out: "mapReduceB"} )
13
14 db.mapReduceB.find()
```

linea 12 Se aplica el MapReduce sobre la colección Ventas.

lineas 2 y 3 Se emiten dos datos, uno que contiene el id del vendedor y otro del comprador, ambos con la calificación correspondiente como valor.

linea 6 Con esto se obtiene una lista por usuario con todas sus calificaciones (tanto de compras como de ventas).

lineas 7 y 8 Se retorna el promedio de las calificaciones (por cada usuario).

3.3 Las operaciones con comisión más alta

```
1  404 NOT FOUND
```

3.4 El monto total facturado por año

```
1  var mapFunctionD = function(){
2    facturaciones = new Array();
3
4    if(this.hasOwnProperty('abonoFijo')){
5      var abono = this.abonoFijo.abono;
6      facturaciones.push(abono);
7    }
8
9    if(this.hasOwnProperty('comisiones')){
10     var comisiones = this.comisiones;
11
12     for (var i = 0; i < comisiones.length; i++) {
13       facturaciones.push(comisiones[i].comision)
14     }
15   }
16
17   emit(this.fecha.slice(this.fecha.length-2, this.fecha.length), facturaciones)
18 };
19
20 var reduceFunctionD = function(year, facturaciones){
21
22   facturacionesFlattened = facturaciones.reduce(function(a, b){return a.concat(b)}, []);
23
24   facturacionAnual = Array.sum(facturacionesFlattened);
25
26   return facturacionAnual;
27 }
28
29 db.Facturas.mapReduce( mapFunctionD, reduceFunctionD, {out: "mapReduceD"} )
30
31 db.mapReduceD.find()
```

linea 29 Se aplica el MapReduce sobre la colección Facturas.

lineas 4 a 7 Si el usuario tiene facturación por abono fijo (i.e. Rubí Oriente) se agrega dicha facturación a la lista de facturaciones.

lineas 9 a 15 Si el usuario tiene una o más facturaciones por comisión se agregan todas ellas a la lista de facturaciones.

linea 17 Se utiliza como clave el año.

linea 20 Con esto se obtiene una lista por año con todas las facturaciones (de comisiones o abonos, de todos los usuarios).

lineas 24 y 26 Se suman todas las facturaciones y se retorna dicha suma.

3.5 El monto total facturado por año de las operaciones pertenecientes a usuarios con suscripciones Rubí Oriente

```
var mapFunctionE = function(){
  facturaciones = new Array();

  if(this.hasOwnProperty('abonoFijo')){
    var abono = this.abonoFijo.abono;
    facturaciones.push(abono);

    if(this.hasOwnProperty('comisiones')){
      var comisiones = this.comisiones;

      for (var i = 0; i < comisiones.length; i++) {
        facturaciones.push(comisiones[i].comision)
      }
    }

    emit(this.fecha.slice(this.fecha.length-2, this.fecha.length), facturaciones)
  }
};

var reduceFunctionE = function(year, facturaciones){

  var facturacionesFlattened = 0;
  facturacionesFlattened = facturaciones.reduce(function(a, b){return a.concat(b)}, []);

  facturacionAnual = Array.sum(facturacionesFlattened);
  return (facturacionAnual);
}

db.Facturas.mapReduce( mapFunctionE, reduceFunctionE, {out: "mapReduceE"} )

db.mapReduceE.find()
```

La funcionalidad es igual al anterior pero en este caso sólo se emiten todas las facturaciones si el usuario posee un abono fijo (i.e. Rubí Oriente).

3.6 El total de las publicaciones por tipo de publicación (productos, servicios, mixtas

```
1  var mapFunctionF = function(){
2    emit(this.tipoDePublicacion, this.id);
3  };
4
5  var reduceFunctionF = function(tipoDePublicacion, publicaciones){
6    return publicaciones.length;
7  }
8
9  db.Publicaciones.mapReduce( mapFunctionF, reduceFunctionF, {out: "mapReduceF"} )
10
11 db.mapReduceF.find()
```

linea 9 Se aplica el MapReduce sobre la colección Publicaciones.

linea 2 Se toma el tipo de publicación como clave, el valor es el id (único para cada publicación).

linea 5 Con esto se obtiene una lista de todas las publicaciones agrupadas por tipo de publicación.

linea 6 Se retorna la cantidad de publicaciones en cada grupo.

4 Sharding

4.1 Introducción

Sharding es la solución que utiliza MongoDB para brindar un acceso a la información almacenada en un tiempo razonable a medida que aumenta el volumen de datos. Permite escalar horizontalmente, es decir agregar nuevas computadoras para distribuir la información. A continuación describimos a grandes rasgos el funcionamiento de este sistema, se evita entrar en ciertos detalles específicos de los algoritmos y procesos involucrados porque no resulta fundamental para entender la solución propuesta.

4.2 Estructura de un sharded cluster

Una colección es dividida en subconjuntos denominados "shards" (se debe configurar un campo del esquema como 'shard key'), cada shard contiene chunks que poseen información en un rango del shard key configurado, de esta manera la información con una shard key similar se mantendrá próxima (en la mayoría de los casos).

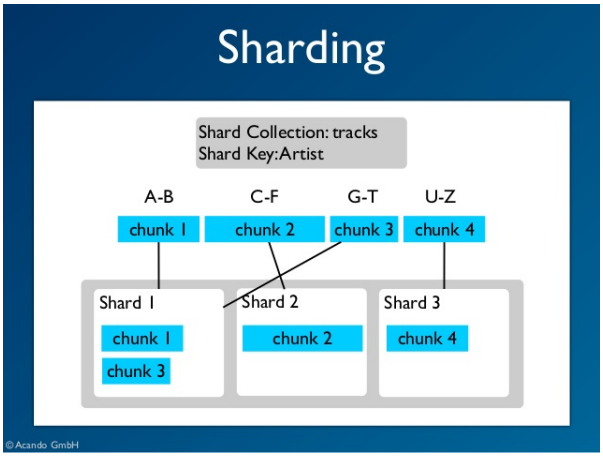

Figure 1: Diagrama de sharding

Por ejemplo, si en una colección se elige 'mes del año' como 'shard key', y tengo tres chunks, para el primer rango puedo tener los datos de Enero a Abril, para el segundo de Abril a Agosto, y el último de Agosto a Diciembre. A medida que se agregue información, puede suceder que se dividan estos chunks, y quede por ejemplo un chunk por mes del año. A partir de ese momento no será posible dividir para formar nuevos rangos, por lo tanto los chunks se vuelven cada vez más grandes.

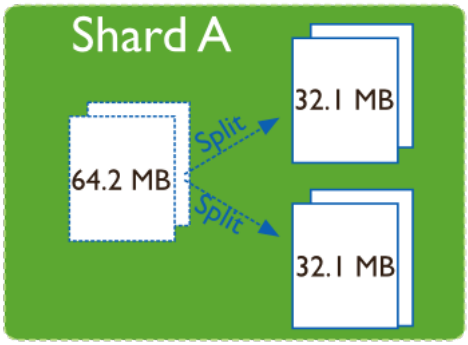

Figure 2: Division de chunk

4.3 Balanceo y migración

Los shards se distribuirán en diferentes servidores, lo deseable es contar con la información distribuida de la mejor forma posible para optimizar las lecturas y escrituras. Además de dividirse, un chunk puede ser migrado de un shard a otro, de esta forma se busca balancear la carga entre los servidores.

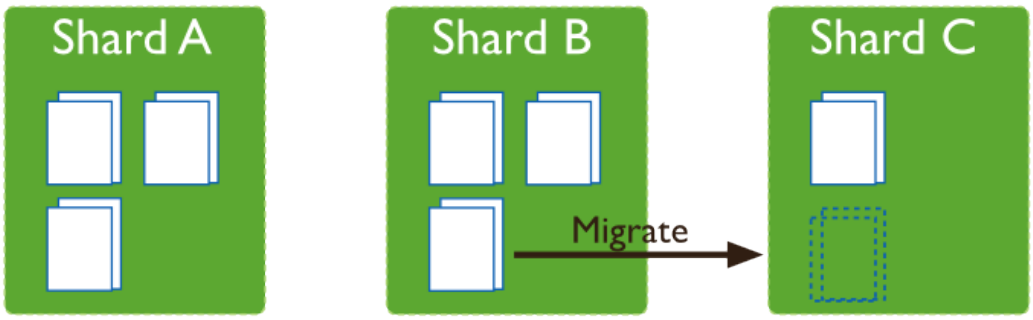

Figure 3: Migración de chunk

4.4 Elección de 'Shard key'

A partir de lo desarrollado, podemos concluir que si queremos utilizar sharding en nuestra colección, resulta fundamental escoger una 'shard key' apropiada. No hay manera de elegir una clave perfecta, para eso se tienen en cuenta una serie de criterios sobre como se comportan los datos que manejamos.

- Cardinalidad: que tanto puede dividirse el rango de valores, lo deseable es tener una cardinalidad alta, es decir que pueda generar gran cantidad de rangos. Un ejemplo de alta cardinalidad sería el DNI, de baja puede ser el código de provincia (en última instancia siempre hay que tener en cuenta el contexto en el que se usan los datos).
- Distribución de las escrituras: lo ideal es distribuir las escrituras de manera uniforme entre los shards, si se escoge una clave que crece de manera monótona, resulta en un exceso de inserts sobre un único shard.
- Distribución de las lecturas: como en el caso de las escrituras, queremos obtener distribución uniforme.
- Tipo de consultas: tener en cuenta que consultas se realizarán sobre la colección, si involucra búsqueda por rango o un documento en particular, en cualquier caso lo deseable es que se accedan la menor cantidad de shards posibles. Para eso es necesario que la shard key esté presente en la consulta efectuada. Por ejemplo si quiero consultar documentos por mes, puedo configurar una 'shard key' compuesta por mes y algún otro campo, de esta forma la información de un mes en particular estará almacenada en un mismo shard (de vuelta depende del caso, pero en general sucedería así).

4.5 Implementación

Se utiliza la arquitectura de prueba

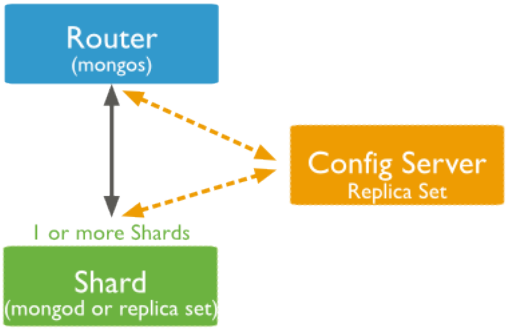

Figure 4: Test Architecture

Trabajando sobre publicaciones con el siguiente esquema.

```
1  {
2  "id": (Integer),
3  "titulo": (String),
```

```
  "tipoDePublicacion": (productos | servicios | mixta),
  "fecha": (Date)
  }
```

Generamos los datos de forma aleatoria(uniformemente) sobre un período de años acotado (5 años), 'tipo de publicación' puede tomar tres valores (servicios, productos y mixta).

Observaciones:

- En los siguientes gráficos, el eje x representa el número de iteración de ingreso de datos, en cada una se insertan 20000 documentos.
- Se decidió representar algunos de los gráficos con barras ya que había mucha colisión entre las lineas, creemos que esta reprensetación ayuda visualmente.

Realizamos la experimentación para tres formas de sharding distintas:

4.5.1 Shard por índice simple: Tipo de publicación

En primera instancia escogimos 'tipo de publicación' como shard key. Obtuvimos una distribución pareja, ya que eran 3 shards para 3 rangos posibles. A simple vista podría parecer una buena clave, pero hay que tener en cuenta que los datos reales podrían tener otra distribución, y la baja cardinalidad en ese caso sería un problema, suponiendo un 90% de publicaciones de artículos implicaría tener los shards totalmente desbalanceados.

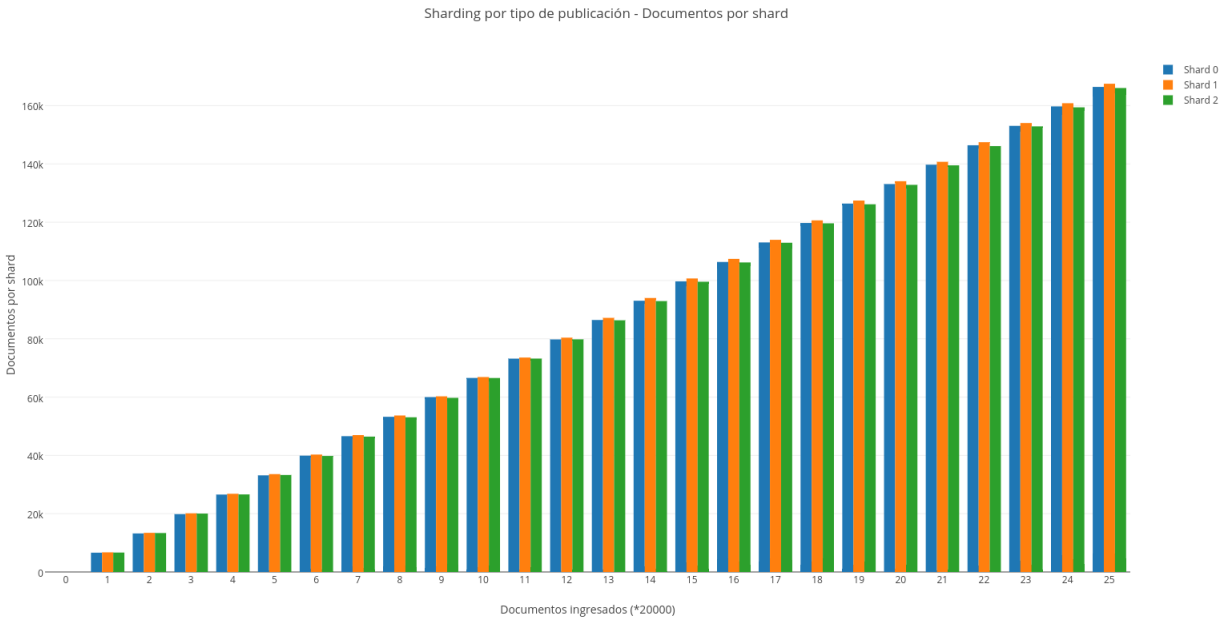

Figure 5: Documentos por Shard

Con los datos obtenidos a través de las pruebas, generamos gráficos a partir de otros parámetros como datos estimados por chunk y datos por shard, los omitimos en el informe porque no agregan información interesante.

Para observar mejor el gráfico se puede acceder mediante el siguiente link a su representación online:

- Documentos por shard

4.5.2 Shard por índice simple: Fecha

Analizando los resultados obtenidos en el punto anterior, nos dimos cuenta que esa clave presentaba muchos inconvenientes y decidimos utilizar otro índice simple, en este caso la fecha, que resultó más interesante. A medida que se insertaron los datos se puede observar como se produce el balanceo de shards a través de la división y migración de los chunks, ya que es una clave con cardinalidad alta. Hay que tener en cuenta que los datos fueron generados de forma aleatoria, quizás en un caso real las fechas serían consecutivas lo que deriva en que las inserciones apunten al mismo shard.

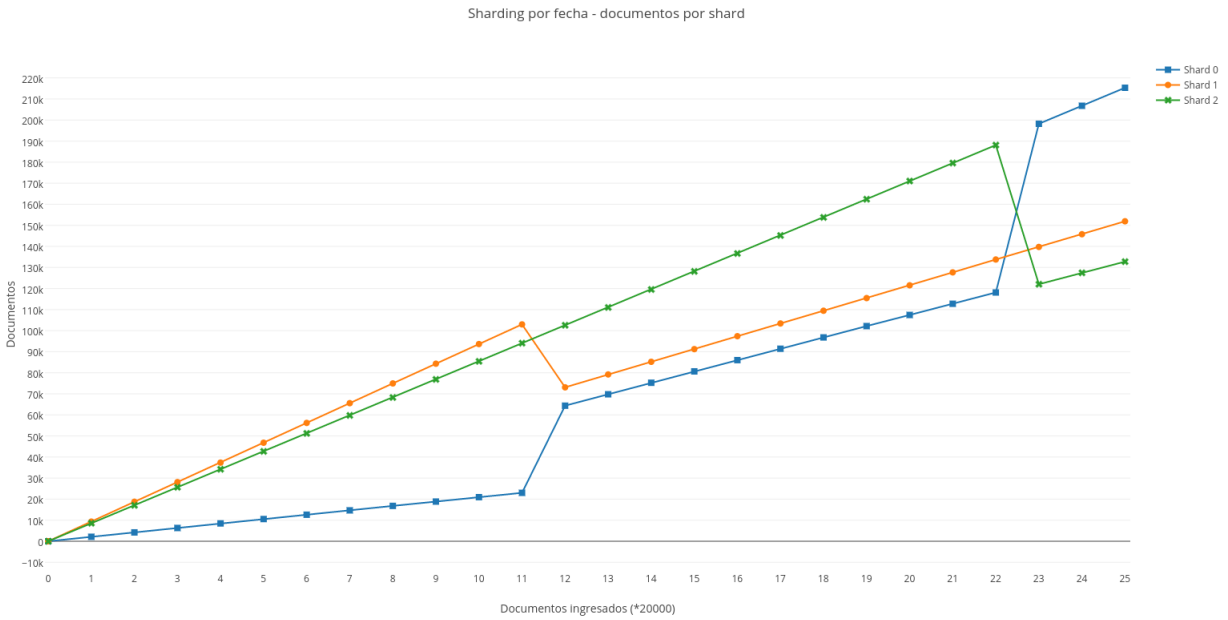

Figure 6: Documentos por shard

Aquí se observa como se distribuye la carga de datos por shard, no es una distribución perfectamente uniforme pero crecen en general al mismo ritmo. Los cambios bruscos se deben a migraciones de datos, posiblemente por un chunk muy grande que fue dividido y luego una parte migrada a otro shard.

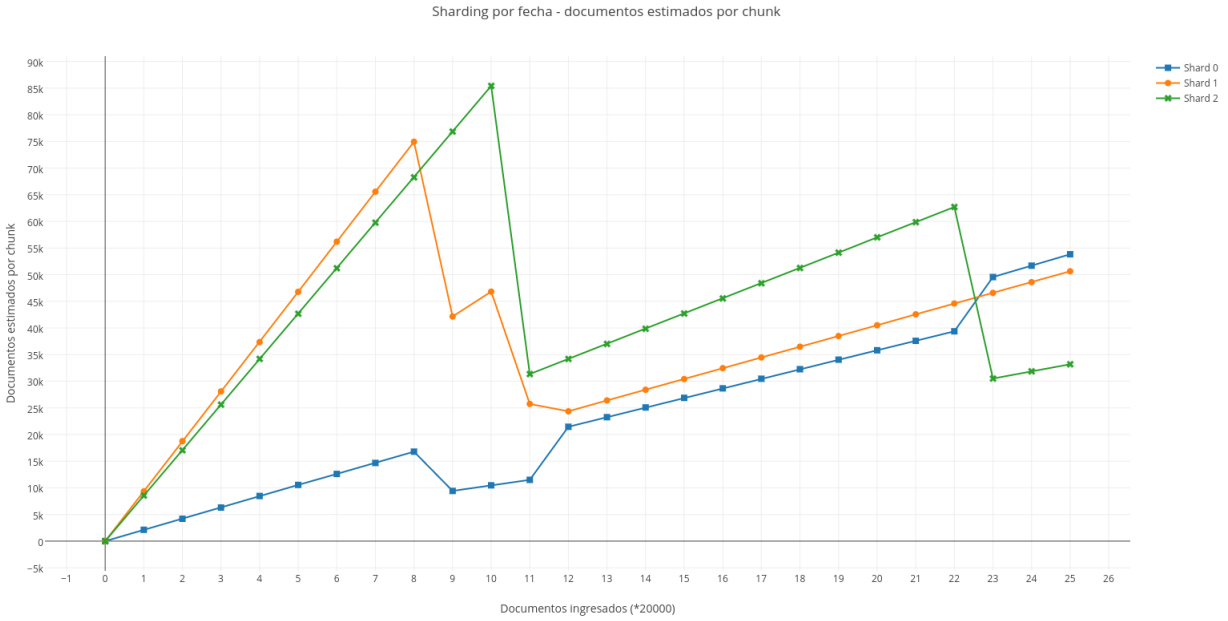

Figure 7: Datos por chunk

En la figura se observa la evolución del promedio de datos por chunk para cada shard, las primeras

iteraciones presentan muchas fluctuaciones, debido a las divisiones y migraciones de chunks, luego se observa como se estabiliza y se mantiene esa tendencia.

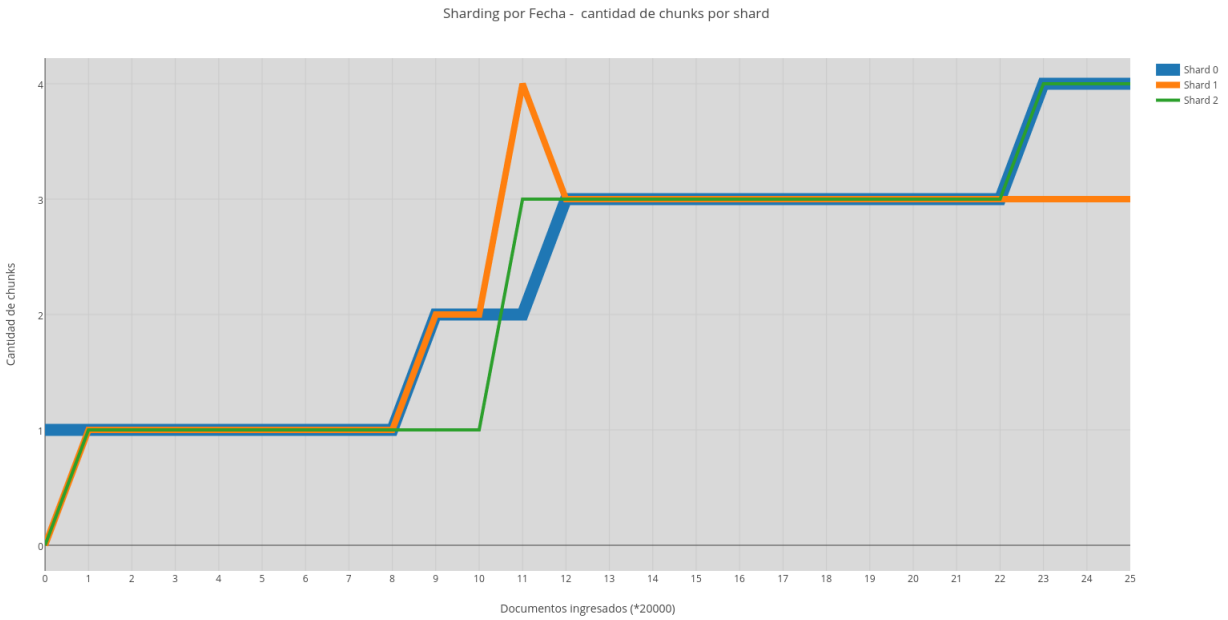

Figure 8: Chunks por shard

Por último se muestra la cantidad de chunks por shard, permitiendo comparar mejor los gráficos anteriores.

Para observar mejor los gráficos se pueden acceder mediante los siguientes links a su representación online:

- Documentos por shard
- Documentos estimados por chunk
- Cantidad de chunks por shard

4.5.3 Shard por Hash: Id

Por último se utiliza un índice hash sobre el campo 'id' que genera Mongo a cada documento. Se eligió este campo ya que, al no poder elegir una clave compuesta, todos los campos generados por nosotros para las versiones de prueba se repetían demasiado (lo que hubiese generado hash parecidos, sino iguales).

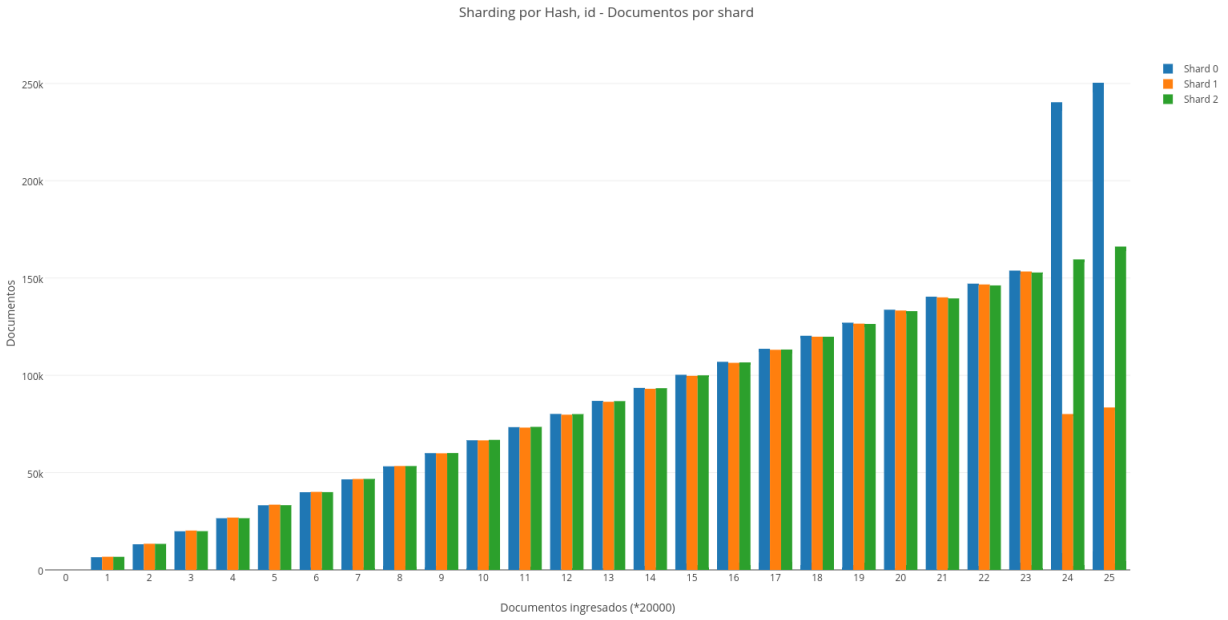

Figure 9: Documentos por shard

Vemos como se distribuyen uniformemente a medida que se ingresan datos. El cambio que se observa en la iteración 24 se puede deber a una mala elección de migración, esto, a medida que los datos crecen, se va corrigiendo. Ya que se espera, de un buen algoritmo de hash, una distribución uniforme.

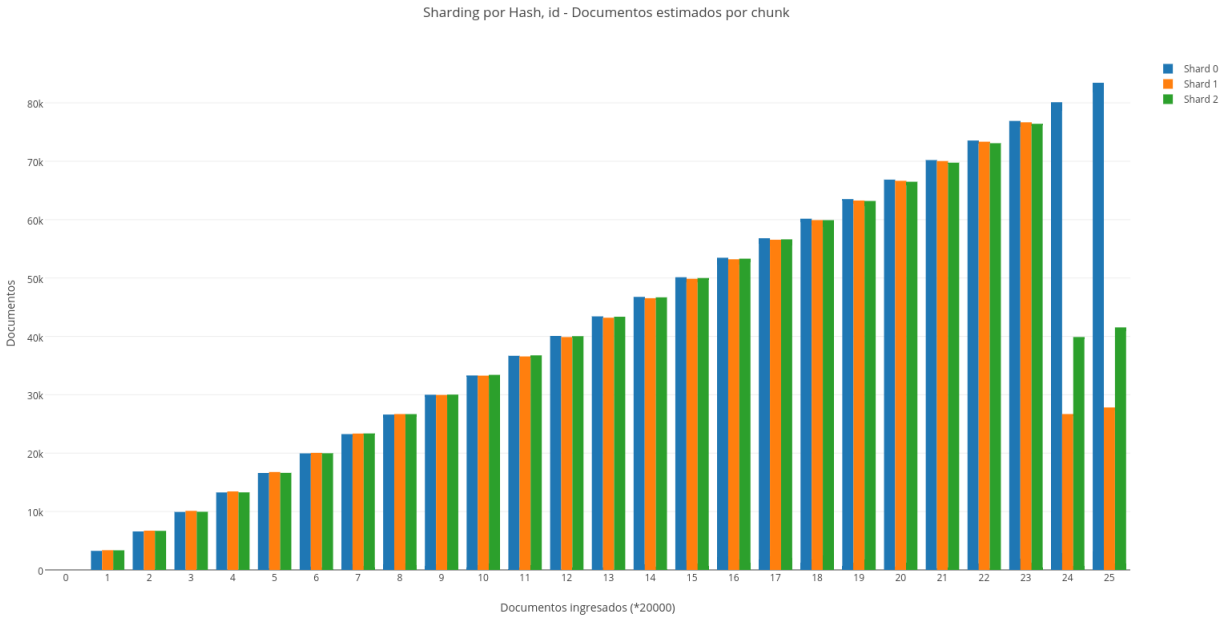

Figure 10: Datos por chunk

Se observa que los chunks quedan desbalanceados en la iteración nro 24. Quedando más cantidad de datos por chunk en el shard 0.

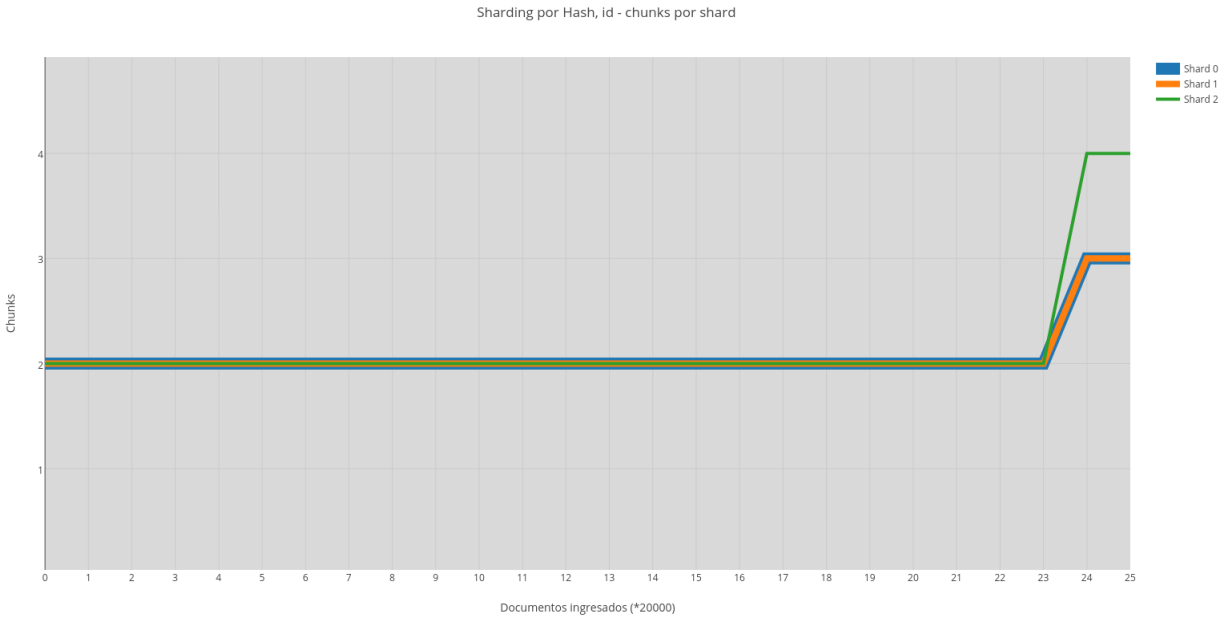

Figure 11: Chunks por shard

Por último se muestra la cantidad de chunks por shard, permitiendo comparar mejor los gráficos anteriores.

Para observar mejor los gráficos se pueden acceder mediante los siguientes links a su representación online:

- Documentos por shard
- Documentos estimados por chunk
- Cantidad de chunks por shard

4.6 Conclusión

Es esencial elegir correctamente el shard key si se quiere una correcta distribución, hay que tener en cuenta distribución de los datos, cantidad de los mismos y cantidad de shards.

Hicimos el primer experimento (tipo de publicación) sin informarnos sobre cómo elegir el shard key, por suerte notamos los problemas a tiempo para repetir el experimento con otra key. Aunque respetando el enunciado no se eligió una shard key compuesta (índice simple). A vista de los resultados una posible clave sería la compuesta por (tipo de publicación, fecha) pero también depende de cómo se quiera acceder a los datos luego.

El índice hash funciona muy bien aunque se pierde la vecinidad de los datos, ya que datos con claves parecidas pueden ir a shards totalmente distintos. Esto puede ser algo tanto bueno como malo. Depende, nuevamente, de cómo se quieran utilizar los datos.

4.7 Fuentes

- https://docs.mongodb.com/v3.0/sharding/
- https://www.mongodb.com/blog/post/on-selecting-a-shard-key-for-mongodb
- http://www.mongodbspain.com/es/tag/shard/